Squirrel: Efficient Synchronized Multi-Signatures from Lattices

Nils Fleischhacker[1⋆], Mark Simkin[2⋆⋆], and Zhenfei Zhang[2⋆⋆⋆]

[1] Ruhr University Bochum
[2] Ethereum Foundation

Abstract. The focus of this work are multi-signatures schemes in the synchronized setting. A multi-signature scheme allows multiple signatures for the same message but from independent signers to be compressed into one short aggregated signature, which allows verifying all of the signatures simultaneously. In the synchronized setting, the signing algorithm takes the current time step as an additional input. It is assumed that no signer signs more than one message per time step and we aim to aggregate signatures for the same message and same time step. This setting is particularly useful in the context of blockchains, where validators are naturally synchronized by the blocks they sign.

We present Squirrel, a concretely efficient lattice-based multi-signature scheme in the synchronized setting that works for a bounded number of 2^τ time steps and allows for aggregating up to ρ signatures at each step, where both τ and ρ are public parameters upon which the efficiency of our scheme depends. Squirrel allows for non-interactive aggregation of independent signatures and is proven secure in the random oracle model in the presence of rogue-key attacks assuming the hardness of the short integer solution problem in a polynomial ring.

We provide a careful analysis of all parameters and show that Squirrel can be instantiated with good concrete efficiency. For $\tau = 24$ and $\rho = 4096$, a signer could sign a new message every 10 seconds for 5 years non-stop. Assuming the signer has a cache of 112 MB, signing takes 68 ms and verification of an aggregated signature takes 36 ms. The size of the public key is 1 KB, the size of an individual signature is 52 KB, and the size of an aggregated signature is 771 KB.

1 Introduction

A multi-signature scheme [IN83, MOR01] allows for compressing multiple signatures for the same message, generated under independent keys, into one short aggregated signature. Given the corresponding public keys, the message, and the aggregated signature, anyone can verify the validity of all signatures simultaneously.

Such signature schemes are particularly useful in the context of cryptocurrencies, where a set of validators maintain a public append-only ledger. The ledger should only contain valid data and minimizing the amount of data stored on the ledger is crucial for the overall efficiency of the cryptocurrency. In regular time intervals, new candidate data blocks appear that may or may not be added to the ledger. If a validator deems a data block eligible for addition to the ledger, they will vouch for it by signing it. If enough validators have signed a specific data block, then it is added to the ledger along with all the signatures vouching for it. In this setting multi-signatures allow for storing less data on the ledger by replacing all individual signatures with the aggregated signature.

It has been shown that multi-signatures can be constructed from a variety of assumptions, such as the RSA assumption [IN83, OO93], discrete logarithm assumptions [BDD+00, MOR01, BN06, BCJ08, NRS21], and pairings-based assumptions [BGLS02, Bol03, LOS+06, BDN18, DGNW20].

⋆ mail@nilsfleischhacker.de. Funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) under Germany's Excellence Strategy - EXC 2092 CASA - 390781972.

⋆⋆ mark.simkin@ethereum.org
⋆⋆⋆ zhenfei.zhang@ethereum.org

Unfortunately, all of the above assumptions are susceptible to quantum attacks [Sho94] and there has been little work in multi-signatures schemes that plausibly remain secure in the presence of a quantum adversary.

A number of recent works [ES16, FH19, MJ19, PD20, KD20, FH20, DOTT21, BTT22] proposed multi-signatures schemes whose security relies on the hardness of lattice assumptions which are currently considered hard even against quantum adversaries. However, none of these are quite suitable for practical depoyment in the envisioned use-case.

All of the proposed schemes require interaction between the independent signers to aggregate the signatures. In applications such as the one sketched above the signers may be online at different times and are potentially not even aware of who the others signers are, thus making interactive aggregation problematic. Ideally, aggregation of signatures should be non-interactive in the sense that it does not require further interaction between any of the signers and the aggregating entity.

The interactive signing protocols of Fukumitsu and Hasegawa [FH19, FH20], Ma and Jiang [MJ19], and Peng and Du [PD20] have a runtime that grows exponentially with the number of participants. This is caused by rejection sampling procedures employed by the underlying lattice signature schemes (e.g. Dilithium). Each user will reject a candidate signature with some probability to prevent information leakage. Rejection by any user requires the entire protocol to restart. This makes the schemes non-applicable with potentially 1000s of signers.

The schemes of El Bansarkhani and Sturm [ES16] as well as Ma and Jiang [MJ19] are only proven secure in a setting where all signing keys, even the adversarial ones, are generated honestly. In reality, an adversary could attempt to perform a so-called rogue key attack, where maliciously formed keys are chosen depending on the honest keys, such that they can forge aggregated signatures that supposedly correspond to a set of keys consisting of both, honest and malicious keys. From a security perspective the aggregated signature should remain unforgeable even if malicious keys are included and aggregation is performed by the adversary. Finally, the scheme of Kansal and Dutta [KD20] was actually shown to be insecure by Liu et al. [LTT20].

The best option among the previous works in this area is the multi-signatures scheme of Boschini, Takahashi, and Tibouchi [BTT22], which provides security against rogue-key attacks. However, it still has the drawback of an interactive aggregation procedure described above.

1.1 Our Contribution

In this work, we focus on multi-signature schemes in the synchronized setting [GR06, AGH10, HW18, DGNW20]. Here, the signing algorithm is given an additional time step t as input along with the message and the secret key. It is assumed that no signer produces more than one signature per time step. Rather than aiming to aggregate any set of signatures we aim to aggregate signatures by independent signers for the same message and same time step. Going back to our previous append-only ledger example, we observe that the validators are naturally synchronized and only aim to aggregate signatures for the same data block which can be associated with a time step t.

We present Squirrel[3], a *concretely efficient* lattice-based multi-signature scheme in the synchronized setting that works for a bounded number of 2^τ time steps and allows for aggregating up to ρ signatures at each step, where both τ and ρ are public parameters upon which the efficiency of our scheme depends. Squirrel allows for non-interactive aggregation of signatures and is secure against rogue key attacks in the random oracle model assuming the hardness of the short integer solution problem in a polynomial ring.

[3] Our construction, just like our rodent friends from the Sciuridae family, heavily rely on (binary) trees.

		Computational			Bandwidth			
		Offline Sign	Online Sign	Verify	sk	pk	σ	σ_{agg}
Asymptotic Efficiency	worst	$\widetilde{\mathcal{O}}(2^\tau)$	$\mathcal{O}(1)$	$\mathcal{O}(\tau)$	$\mathcal{O}(\lambda)$	$\widetilde{\mathcal{O}}(n)$	$\widetilde{\mathcal{O}}(\tau n)$	$\widetilde{\mathcal{O}}(\tau n)$
	average	$\widetilde{\mathcal{O}}(1)$						
Concrete Efficiency		0.4 s	2.3 ms	36 ms	8 MB	1 KB	52 KB	771 KB
		25 ms			128 MB			
		1.6 ms			2 GB			

Table 1. The asymptotic worst-case and average-case along with concrete worst-case costs of Squirrel. Here λ denotes the security parameter and ρ the maximum number of signatures that can be aggregated. The maximum number of signatures that can be issued under one key pair is 2^τ. The column σ specifies the size of an individual signature while σ_{agg} specifies the size of an aggregated signature. Asymptotic worst-case cost is measured in terms of ring multiplications. The $\widetilde{\mathcal{O}}(\cdot)$ notation hides logarithmic dependencies. Concrete costs are measured for $\tau = 24$ and $\rho = 4096$ with $\lambda = 112$.

It may seem that having an upper bound on the number of signatures is a severe restriction that limits the practical usefulness of our results. To see that this is not the case in many settings, we note that even with a τ as small as 24, a single signing key supports signing a new message every 10 seconds for 5 years non-stop.

Squirrel is both asymptotically and concretely efficient in most parameters as can be seen in Table 1. Keys and signature sizes are reasonably small and verification of an aggregated signature only takes a few tens of milliseconds. The main (theoretical) bottleneck of Squirrel is the asymptotic worst-case signing cost. Fortunately, our construction possesses several nice features that alleviate the asymptotic inefficiency in practice. Our construction is an online/offline signature scheme [EGM90] which means that the majority of the computational cost of the signing procedure can be preprocessed before the message to be signed is known. The amortized overall computational cost per signature is exponentially smaller than the worst-case cost, which means that signing is computationally cheap most of the time and only rarely requires a larger computational effort. Lastly, we show that the concrete computational worst-case costs of the signing procedure can be significantly reduced by storing somewhat larger secret keys. As an exemplary data point, one can see in Table 1 that a 2 GB secret key allows for a total signing time, i.e. offline plus online signing times, of below 4 ms. We stress that storing such a "large" secret key with modern hardware does not pose a problem in the absolute majority of use-cases, for instance, where signers are blockchain validators with adequate resources. Verification of the aggregated signature, which is 771 KB large, only takes 36 ms.

A naive construction of a lattice-based multi-signature scheme with non-interactive aggregation is to simply append individual signatures of a plain lattice-based signature scheme. Such a scheme can, for instance, be instantiated with Dilithium signatures [DKL+18] or Falcon signatures [PFH+20]. When comparing such a solution to ours, for the parameters from above, the naive scheme with Dilithium requires roughly 3 KB per signature and 0.2 ms per verification. For 4096 aggregated signatures, the size would be around 12 MB and verification would take around 800 ms. In comparison, our solution requires 771 KB for the aggregated signature, reducing size by 94%, and verification is faster by a factor of 20. For Falcon signatures, we observe smaller gains, reducing 71% in size, and accelerating verification by a factor of 4. In terms of verification times, our result is also on-par with pre-quantum algorithms, since it takes roughly 200 ms to verify 4096 ECDSA signatures [Lib22] and 2 ms to verify 4096 BLS signatures [Sup22]. When there are more signatures

to aggregate, our benchmark shows that our signature size scales sub-logarithmically with regard to ρ. We provide a detailed discussion of the concrete efficiency of our scheme in Section 6.

1.2 Real-World Impact

Squirrel can be used in the context of major cryptocurrencies, such as Ethereum 2 and DFinity. In a nutshell, both these systems are keeping track of a continuously growing ordered chain of data blocks in a distributed manner. To ensure that no malformed blocks are added to the chain, each block has to include a sufficient number of signatures that vouch for their validity. Both of the mentioned cryptocurrencies are currently relying on the quantum-insecure BLS multi-signature scheme [BLS01] to compress the signatures in each block. For more details, we refer the interested reader to Sections 5.7 and 5.8 in the DFinity whitepaper[4] or the annotated Ethereum 2 specification[5]. Constructing plausibly quantum-secure alternatives to BLS signatures with good concrete efficiency has so far been a tough nut to crack.

To understand why Squirrel can be used in the context of these cryptocurrencies we need to make two crucial observations. Firstly, the signatures we aim to aggregate are naturally synchronized by the length of the current chain, meaning that signatures for block i can be associated with a time step i. Secondly, both cryptocurrency designs enforce that no validator can vouch for more than one data block at any point in time. These two restrictions on how multi-signatures are being used here perfectly match the two restrictions our construction has.

1.3 Limitations

Since Squirrel is proven secure under the assumed hardness of the short integer solution problem in a polynomial ring, it does not directly fall victim to attacks by a quantum adversary. However, our security proof relies on a variant [BN06] of the forking lemma [PS96], and therefore uses a rewinding strategy that *does not apply* to quantum algorithms. Although it is *plausible* that our scheme is secure against quantum attackers, we do not currently know how to prove this and leave such a proof as an open question. In this context, it may be noteworthy, that a proof of security against *quantum* attackers in the *classical* random oracle model would be sufficient, because such a proof could be lifted to the *quantum* random oracle model [BDF+11] using the work of Yamakawa and Zhandry [YZ21].

1.4 Technical Overview

Let us start with a very simple solution. Assume we are already given a one-time multi-signature scheme, i.e. a scheme, where a signer can sign exactly once under a given public key. To create a signature scheme that allows for signing 2^τ many times, a signer can generate 2^τ many independent one-time signature key pairs and publish a public key, which is the concatenation of all one-time public keys. To sign at time t, the signer signs using the t-th secret key. Such a scheme would already constitute a valid multi-signature solution in the synchronized setting for a bounded number of signatures. The main drawback of this approach is that the public key grows linearly in 2^τ, which is completely unacceptable.

[4] https://dfinity.org/whitepaper.pdf
[5] https://github.com/ethereum/annotated-spec/blob/master/phase0/beacon-chain.md#attestation

As a subsequent iteration of the idea above, one can attempt to publish the root node v of a Merkle tree that is computed on top of all the public keys. The tree serves as a commitment to the vector of individual public keys. To sign a message at time t, we would now publish a signature under the t-th key pair along with the t-th public key and a membership proof, which shows that the key is indeed the t-th leaf of the tree with the root node v. The problem is that this solution breaks the aggregation property, since one-time signatures can still be aggregated, but the membership proofs of the separate Merkle trees cannot.

Luckily for us, the idea of Merkle trees with homomorphic properties has already been studied by Papamanthou et al. [PSTY13]. In principle, their construction of a "homomorphic Merkle tree" is sufficient to make the simple idea from above work. Using these trees, one can now aggregate both the one-time multi-signatures and the membership proofs. To make this solution secure against rogue-key attacks, we can not just sum up separate signatures, but instead compute a random linear combination thereof, where the weights are chosen via a random oracle.

The main issue with the work of Papamanthou et al. [PSTY13] is the large asymptotic and concrete costs associated with their tree construction. When trying to realize our approach with their work, one obtains signatures sizes in the gigabyte range which would be prohibitively expensive for practical scenarios. On a very high level the main issue, among others, with their construction is that their security relies on a lattice-based assumption where the parameters grow linearly in the number of leaves of their tree. The parameters of the used assumption further deteriorate, when the random weights are applied to the membership proofs.

A simpler and more efficient set membership data structure from lattices was considered by Libert et al. [LLNW16]. Whereas Papamanthou et al. compute the labels of internal nodes as weighted sums of all leaves rooted in that node, the construction of Libert et al. is essentially a standard Merkle tree instantiated with Ajtai's hash function [Ajt99] with an additional decomposition step to map hash values into the domain of the hash function. Their work did not need or consider any homomorphic properties that their tree might possess.

In this work, we observe that the construction of Libert et al. does indeed have the homomorphic properties that we need for our application, but unfortunately does not allow for efficiently aggregating random linear combinations of authentication paths from different trees. The tree of Libert et al. works with values over $\mathbb{Z}_q$ that are required to have small norm, when interpreted as integers. For our random linear combinations, however, we need to choose weights that come from a super-polynomially large set. Such a large subset of $\mathbb{Z}$ will necessarily have elements with a superpolynomially large norm, which would result in a blow-up in the asymptotic and concrete sizes of the aggregated paths in Libert et al.'s construction.

In this paper, we present a new construction of such a Merkle tree with homomorphic properties that does not have the drawbacks of the previous works and is concretely efficient. Essentially, we describe an analogue of the tree of Libert et al. instantiated over a polynomial ring, where superpolynomially large subsets of elements with small norm exist. Additionally the construction is made more efficient by using a seperate hash function with a wider input for the leaf layer.

We present an appropriate one-time multi-signature scheme that works well in combination with our tree as outlined above. We stress that even though our construction is simple on a conceptual level, realizing the idea and making it concretely efficient is far from it.

Paper Outline. We define some notation and review some existing definitions that will be used throughout the paper in Section 2. We formally define our notion of a homomorphic vector commit-

ment and show how to instantiate it with a construction that resembles a Merkle tree in Section 3. We define the notion of a one-time multi-signature scheme that we need and instantiate it in Section 4. Our multi-signature scheme is presented in Section 5. Finally, we discuss all relevant concrete parameters and provide extensive benchmarks of our construction in Section 6.

2 Preliminaries

This section introduces notation, some basic definitions and lemmas that we will use throughout this work. We denote by $\lambda \in \mathbb{N}$ the security parameter and by $\mathsf{poly}(\lambda)$ any function that is bounded by a polynomial in λ. A function f in λ is negligible, if for every $c \in \mathbb{N}$, there exists some $N \in \mathbb{N}$, such that for all $\lambda > N$ it holds that $f(\lambda) < 1/\lambda^c$. We denote by $\mathsf{negl}(\lambda)$ any negligible function. An algorithm is PPT if it is modeled by a probabilistic Turing machine with a running time bounded by $\mathsf{poly}(\lambda)$.

Let X be a set. We write $x \leftarrow X$ to denote the process of sampling an element of X uniformly at random. Let $n \in \mathbb{N}$, we denote by $[n]$ the set $\{0, \ldots, n\}$. Let T be a full binary tree of depth d. We denote the root node of T by the empty string ϵ, and for any node v, $v\|0$ and $v\|1$ denotes the left and right child of v respectively. In particular, $\{0,1\}^d$ is the set of leaves of T. A labeled full binary tree with labels in X is represented by a labeling function $\mathsf{label} : \{0,1\}^{\leq d} \to X$.

Let $\boldsymbol{v}$ be a vector. We write $\boldsymbol{v}^\intercal$ to denote its transpose and v_i to denote the i-th entry in the vector for $i \in [|\boldsymbol{v}| - 1]$. Further, $\boldsymbol{v}_{<i}$ denotes the i-length prefix of $\boldsymbol{v}$. Similarly for a bit-string s, s_i denotes the i-th bit of s and $s_{<i}$ denotes the prefix consisting of the first i bits of s. Note that vectors and bit-strings are zero-indexed. From time to time we will slightly abuse this notation and use a bit-string s as an index. In this case the index is to be understood as the canonical interpretation of s as an integer in little-endian encoding.

Without loss of generality, we work on a power-of-two cyclotomic polynomial ring. Let $\Phi_{2n} = x^n + 1$ the cyclotomic polynomial with n a power of 2. We work in a polynomial ring $\mathcal{R} = \mathbb{Z}[x]/\langle x^n + 1\rangle$ and represent elements of $\mathcal{R}$ as n-dimensional vectors $\mathbb{Z}^n$ with $(c_0, \ldots, c_{n-1})^\intercal \in \mathbb{Z}^n$ representing the ring element $\sum_{i=0}^{n-1} x^i \cdot c_i$. Let q be some prime such that $q \equiv 1 \bmod 2n$. $\mathcal{R}_q$ refers to the subset of $\mathcal{R}$ represented by vectors in $\mathbb{Z}_q^n$. Let $x = \boldsymbol{c} \in \mathcal{R}$ be a ring element. We denote $\|x\| = \|\boldsymbol{c}\|_\infty = \max_{i\in[n-1]}|c_i|$ and $\|x\|_1 = \|\boldsymbol{c}\|_1 = \sum_{i\in[n-1]}|x_i|$. For an element $a \in \mathcal{R}_q$ we denote by $\|a\|$ or $\|a\|_1$ the respective norm over $\mathcal{R}$.

We denote by $\mathcal{B}_\beta$ the ball $\mathcal{B}_\beta = \{a \in \mathcal{R}_q \mid \|a\| \leq \beta\}$ and by $\mathcal{T}_\alpha = \{a = (a_0 + a_1 \cdot x + \cdots + a_{n-1}x^{n-1}) \in \mathcal{R} \mid \|a\| = 1 \wedge \sum_{i=0}^{n-1} |a_i| = \alpha\}$ the set of polynomials with ternary coefficients, i.e. coefficients from $\{-1, 0, 1\}$, with exactly α non-zero coefficients. The following simple lemma allows us to bound the norm of the product of two polynomials.

Lemma 1 ([Mic07]). *Let $a, b \in \mathcal{R}$ be two polynomials. Then $\|b \cdot a\| \leq \|a\|_1 \cdot \|b\|$.*

The computationally hard problem upon which the security of our constructions relies is the short integer solution problem defined over rings as follows.

Definition 1 (**Ring Short Integer Solution Problem**). *For a ring $\mathcal{R}$ and parameters $\mu, q, \beta \in \mathbb{N}$, the $\mathsf{SIS}_{\mathcal{R},q,\mu,\beta}$ problem is hard if for all PPT algorithms $\mathcal{A}$ it holds that*

$$\Pr[\boldsymbol{a} \leftarrow \mathcal{R}_q^\mu; \boldsymbol{s} \leftarrow \mathcal{A}(\boldsymbol{a}) : \boldsymbol{s} \in \mathcal{B}_\beta^\mu \setminus \{\boldsymbol{0}\} \wedge \boldsymbol{a}^\intercal \boldsymbol{s} = 0] \leq \mathsf{negl}(\lambda)$$

We will be using a minor variation of the general forking lemma as introduced by Bellare and Neven [BN06] that explicitly deals with oracle algorithms.

Lemma 2 (General Forking Lemma). *Fix an integer $p \geq 1$ and a set H of size $h \geq 2$. Let $\mathcal{A}$ be a randomized oracle algorithm with randomness space R that on input $x, h_0, \ldots, h_{p-1}$ and given access to an oracle $\mathcal{O}$ returns a triple, the first element of which is a bit, the second element of which is an integer in the range $0, \ldots, p-1$ and the third element of which we refer to as a side output. Let IG be a randomized algorithm that we call the input generator. The accepting probability of $\mathcal{A}$, denoted ϵ, is defined as*

$$\epsilon := \Pr\left[\begin{array}{r} (x,y) \leftarrow \mathsf{IG}; \\ h_0, \ldots, h_{p-1} \leftarrow H; : b = 1 \\ (b,i,\sigma) \leftarrow \mathcal{A}^{\mathcal{O}(y,\cdot)}(x, h_0, \ldots, h_{p-1}) \end{array}\right]$$

The forking algorithm $\mathsf{F}_{\mathcal{A}}$ associated with $\mathcal{A}$ is the randomized oracle algorithm that takes input x, is given access to an oracle $\mathcal{O}$ and proceeds as follows:

$\mathsf{F}_{\mathcal{A}}^{\mathcal{O}(y,\cdot)}(x)$

$r \leftarrow R$
$h_0, \ldots, h_{p-1} \leftarrow H$
$(b, i, \sigma) \leftarrow \mathcal{A}^{\mathcal{O}(y,\cdot)}(x, h_0, \ldots, h_{p-1}; r)$
if $b = 0$
  return $(0, \perp, \perp)$
$h'_i, \ldots, h'_{p-1} \leftarrow H$
$(b', i', \sigma') \leftarrow \mathcal{A}^{\mathcal{O}(y,\cdot)}(x, h_1, \ldots, h_{i-1}, h'_i, \ldots, h'_{p-1}; r)$
if $b' = 1$ **and** $i = i'$ **and** $h'_i \neq h_i$
  return $(1, \sigma, \sigma')$
else
  return $(0, \perp, \perp)$

Then

$$\epsilon \leq \frac{p}{h} + \sqrt{p \cdot \Pr[(x,y) \leftarrow \mathsf{IG}; (b, \sigma, \sigma') \leftarrow \mathsf{F}_{\mathcal{A}}^{\mathcal{O}(y,\cdot)}(x) : b = 1]}.$$

3 Homomorphic Vector Commitment

In this section, we formally define the notion of a homomorphic vector commitment that we will need in our main construction. This primitive, on an intuitive level, allows for committing to a long vector by publishing a short commitment value. Individual positions of the vector can then be opened individually with short openings. The commitment scheme should be homomorphic, meaning that a linear combination of individual commitments different vectors be opened to the linear combination of the entries of the individual vectors.

Definition 2. *Let $\mathcal{R}$ be a ring and let $q = q(\lambda) \in \mathbb{N}$. A homomorphic vector commitment scheme (HVC) for domain $\mathcal{R}_q^{\ell_{\mathsf{dom}}}$ is defined by four PPT algorithms* $(\mathsf{Setup}, \mathsf{Com}, \mathsf{Open}, \mathsf{Vrfy})$.

$\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda, \tau)$ *The setup algorithm takes as input the security parameter and the binary logarithm of the length of the committed vectors and outputs public parameters.*

$c \leftarrow \mathsf{Com}(\mathsf{pp}, \boldsymbol{m})$ *The commitment algorithm gets as input the public parameters and a vector $\boldsymbol{m} \in (\mathcal{R}_q^{\ell_{\mathsf{dom}}})^{2^\tau}$ and outputs a commitment $c \in \mathcal{R}_q^{\ell_{\mathsf{com}}}$.*

$d \leftarrow \mathsf{Open}(\mathsf{pp}, c, \boldsymbol{m}, t)$ *The opening algorithm gets as input the public parameters, a commitment, the committed vector, and an index and outputs a decommitment* $d \in \mathcal{R}_q^{\ell_{\mathsf{dec}}}$.

$\boldsymbol{m}/\perp \leftarrow \mathsf{wVrfy}(\mathsf{pp}, c, t, d)$ *The weak verification algorithm takes as input public parameters, a commitment, an index, and a decommitment and outputs either* $\boldsymbol{m} \in \mathcal{R}_q^{\ell_{\mathsf{dom}}}$ *or an error symbol.*

$\boldsymbol{m}/\perp \leftarrow \mathsf{sVrfy}(\mathsf{pp}, c, t, d)$ *The strong verification algorithm takes as input public parameters, a commitment, an index, and a decommitment and outputs either* $\boldsymbol{m} \in \mathcal{R}_q^{\ell_{\mathsf{dom}}}$ *or an error symbol.*

Let $\rho \in \mathbb{N}$ *and* $W \subseteq \mathcal{R}$. *A vector commitment is* (ρ, W)*-homomorphically correct, if for all security parameters* $\lambda \in \mathbb{N}$, *vector lengths* $2^\tau = \mathsf{poly}(\lambda)$, $\ell \in [\rho]$, *vectors* $\boldsymbol{m}^0, \ldots, \boldsymbol{m}^{\ell-1} \in (\mathcal{R}_q^{\ell_{\mathsf{dom}}})^{2^\tau}$, *ring elements* $w^0, \ldots, w^{\ell-1} \in W$, *and indices* $t \in [2^\tau - 1]$ *it holds that*

$$\Pr\left[\begin{array}{l}\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda, \tau);\\ \boldsymbol{c}^i \leftarrow \mathsf{Com}(\mathsf{pp}, \boldsymbol{m}^i);\\ \boldsymbol{d}^i \leftarrow \mathsf{Open}(\mathsf{pp}, \boldsymbol{c}^i, \boldsymbol{m}^i, t)\end{array} : \mathsf{sVrfy}\Big(\mathsf{pp}, \sum_{i=0}^{\ell-1} w^i \cdot \boldsymbol{c}^i, t, \sum_{i=0}^{\ell-1} w^i \cdot \boldsymbol{d}^i\Big) = \sum_{i=0}^{\ell-1} w^i \cdot \boldsymbol{m}_t^i\right] = 1$$

Remark 1. Note that the homomorphic correctness definition above implies regular correctness of unaggregated commitments with $\ell = 1$ and $1 \in W$.

Definition 3 (Position-Binding). *An HVC is position binding if for all security parameters* λ *and all PPT algorithms* $\mathcal{A}$ *it holds that*

$$\Pr\left[\begin{array}{l}\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda, \tau);\\ (c, t, d_0, d_1) \leftarrow \mathcal{A}(\mathsf{pp});\\ \boldsymbol{m}_0 \leftarrow \mathsf{wVrfy}(\mathsf{pp}, c, t, d_0);\\ \boldsymbol{m}_1 \leftarrow \mathsf{wVrfy}(\mathsf{pp}, c, t, d_1)\end{array} : \begin{array}{l}\boldsymbol{m}_0 \neq \boldsymbol{m}_1\\ \wedge \perp \notin \{\boldsymbol{m}_0, \boldsymbol{m}_1\}\end{array}\right] \leq \mathsf{negl}(\lambda).$$

We require that a limited homomorphism holds, even for malicious commitments. For any two, even malicious, commitments and their two respective openings that *strongly* verify, their difference will still weakly verify.

Definition 4. *Let* HVC *be a vector commitment scheme (HVC) for domain* $\mathcal{R}_q^{\ell_{\mathsf{dom}}}$ *with commitment length* ℓ_{com} *and decommitment length* ℓ_{dec}. HVC *is robustly homomorphic if for all security parameters* $\lambda \in \mathbb{N}$, *vector lengths* $2^\tau = \mathsf{poly}(\lambda)$, *public parameters* $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda, \tau)$, *indices* $t \in [2^\tau - 1]$, *(possibly malformed) commitments* $\boldsymbol{c}^0, \boldsymbol{c}^1 \in \mathcal{R}_q^{\ell_{\mathsf{com}}}$, *and (possibly malformed) decommitments* $\boldsymbol{d}^0, \boldsymbol{d}^1 \in \mathcal{R}_q^{\ell_{\mathsf{dec}}}$ *such that*

$$\mathsf{sVrfy}(\mathsf{pp}, \boldsymbol{c}^0, t, \boldsymbol{d}^0) = \boldsymbol{m}^0 \quad \textit{and} \quad \mathsf{sVrfy}(\mathsf{pp}, \boldsymbol{c}^1, t, \boldsymbol{d}^1) = \boldsymbol{m}^1$$

with $\boldsymbol{m}^0, \boldsymbol{m}^1 \neq \perp$ *it holds that*

$$\mathsf{wVrfy}(\mathsf{pp}, \boldsymbol{c}^0 - \boldsymbol{c}^1, t, \boldsymbol{d}^0 - \boldsymbol{d}^1) = \boldsymbol{m}^0 - \boldsymbol{m}^1.$$

Strong vs Weak Verification. A noticable and potentially unusual feature of the above definitions is that it uses two seperate verification algorithms. We note that weak and strong verification *can* be identical, but the definition above is more general and in fact necessary to allow for our lattice based instantiation. To see why, consider the following. Ideally, in a definition featuring only a single verification algorithm, a robust homomorphism would guarantee that for any two *valid* commitment,

decommitment pairs $(\boldsymbol{c}^0, \boldsymbol{d}^0)$, $(\boldsymbol{c}^1, \boldsymbol{d}^1)$ opening to $\boldsymbol{m}^0$ and $\boldsymbol{m}^1$ respectively, $(\boldsymbol{c}^0 - \boldsymbol{c}^1, \boldsymbol{d}^0 - \boldsymbol{d}^1)$ is also valid and opens to $\boldsymbol{m}^0 - \boldsymbol{m}^1$. However, this is inherently difficult to achieve with lattices. In any SIS based construction, the verification must involve checking a bound on the norm of the commitment/decommitment. (The same applies with LWE based constructions and the size of the error.) If the norms of $(\boldsymbol{c}^0, \boldsymbol{d}^0)$ and $(\boldsymbol{c}^1, \boldsymbol{d}^1)$ are already close to but still smaller than the enforced norm-bound, the norm of $(\boldsymbol{c}^0 - \boldsymbol{c}^1, \boldsymbol{d}^0 - \boldsymbol{d}^1)$ will often exceed the bound. This would make the individual pairs valid but their difference invalid, breaking the robust homomorphism. The issue can be sidestepped by using two separate bounds. A smaller bound that is used for correctness and a greater bound that is only used in the security definition. To still allow for a clean abstraction, we encapsulate this in strong and weak verification procedures.

3.1 Homomorphic Vector Commitment for $\mathcal{R}_q$

Having formally defined the primitive we want, we now show how to construct it. We first focus on constructing a vector commitment with domain $\mathcal{R}_q$. In Section 3.2 we will show how to leverage this into a more general construction for domain $\mathcal{R}_q^\xi$.

Our construction is essentially a ring version of a tree construction already presented by Libert et al. [LLNW16] that follows the blueprint initially presented by Papamanthou et al. [PSTY13]. We instantiate the homomorphic vector commitments by constructing a Merkle tree with a "sufficiently" homomorphic hash functions at the internal nodes. The hash function will have different input and output domains and for that reason we will need to apply a decomposition function on the hash outputs at the internal nodes before they can be used as inputs in the computation of the parent nodes' values.

The construction differs from the work of Libert et al. because we require somewhat different properties, in particular the ability to compute random linear combinations of decommitments without blowing up the size. This is achieved by working over an appropriate polynomial ring that allows for a superpolynomially large set of low norm weights. We also take care to adapt the decomposition function to optimize the concrete efficiency of our final construction.

We now define a decomposition function that allows us to map a ring element with possibly large norm to a vector of low norm ring elements and we show that this function has nice homomorphic properties.

Definition 5 (Binary decomposition of $\mathcal{R}_q$ elements). *For any $a = \sum_{i=0}^{n-1} a_i \cdot x^i \in \mathcal{R}_q$, denote by $(a_{i,0}, \ldots, a_{i,\lceil \log q \rceil - 1})^\intercal \in \{0,1\}^{\lceil \log q \rceil}$ the binary decomposition of a_i, i.e.,*

$$a_i := \sum_{j=0}^{\lceil \log q \rceil - 1} a_{i,j} \cdot 2^j.$$

We define the following decomposition of a into binary polynomials:

$$\mathsf{bin}_q : \mathcal{R}_q \to \mathcal{R}_q^{\lceil \log q \rceil}, \quad \mathsf{bin}_q(a) = \left(\sum_{i=0}^{n-1} a_{i,0} \cdot x^i, \ldots, \sum_{i=0}^{n-1} a_{i,\lceil \log q \rceil - 1} \cdot x^i \right).$$

Definition 6 (Projection onto $\mathcal{R}_q$ elements). *For any $\boldsymbol{b} \in \mathcal{R}_q^{\lceil \log q \rceil}$ we define the function*

$$\mathsf{proj}_q : \mathcal{R}_q^{\lceil \log q \rceil} \to \mathcal{R}_q, \quad \mathsf{proj}(\boldsymbol{b}) = \sum_{j=0}^{\lceil \log q \rceil - 1} 2^j \cdot b_j.$$

For the sake of readability we will omit q and simply write bin and proj whenever the modulus is clear from context.

The following two simple lemmas effectively states that the projection function is the inverse of the decomposition function and that the projection function is linear.

Lemma 3. *For all* $a = \sum_{i=0}^{n-1} a_i \cdot x^i \in \mathcal{R}_q$*, it holds that* $\mathsf{proj}(\mathsf{bin}(a)) = a$.

Proof.

$$\begin{aligned}
\mathsf{proj}(\mathsf{bin}(a)) =& \mathsf{proj}\left(\sum_{i=0}^{n-1} a_{i,0} \cdot x^i, \ldots, \sum_{i=0}^{n-1} a_{i,\lceil \log q \rceil - 1} \cdot x^i\right) \\
=& \sum_{j=0}^{\lceil \log q \rceil - 1} 2^j \cdot \sum_{i=0}^{n-1} a_{i,j} \cdot x^i \\
=& \sum_{i=0}^{n-1} x^i \cdot \sum_{j=0}^{\lceil \log q \rceil - 1} 2^j \cdot a_{i,j} \\
=& \sum_{i=0}^{n-1} x^i \cdot a_i = a
\end{aligned}$$

□

Lemma 4. *The projection function* proj *is linear, i.e., for any* $\boldsymbol{b}^0, \boldsymbol{b}^1 \in \mathcal{R}_q^{\lceil \log q \rceil}$ *and any* $w^0, w^1 \in \mathcal{R}_q$*,* $\mathsf{proj}(w^0 \cdot \boldsymbol{b}^0 + w^1 \cdot \boldsymbol{b}^1) = w^0 \cdot \mathsf{proj}(\boldsymbol{b}^0) + w^1 \cdot \mathsf{proj}(\boldsymbol{b}^1)$.

Proof.

$$\begin{aligned}
\mathsf{proj}(w^0 \cdot \boldsymbol{b}^0 + w^1 \cdot \boldsymbol{b}^1) =& \sum_{j=0}^{\lceil \log q \rceil - 1} 2^j \cdot (w^0 \cdot b_j^0 + w^1 \cdot b_j^1) && \text{(Definition 6)} \\
=& w^0 \cdot \left(\sum_{j=0}^{\lceil \log q \rceil - 1} 2^j \cdot b_j^0\right) + w^1 \cdot \left(\sum_{j=0}^{\lceil \log q \rceil - 1} 2^j \cdot b_j^1\right) \\
=& w^0 \cdot \mathsf{proj}(\boldsymbol{b}^0) + w^1 \cdot \mathsf{proj}(\boldsymbol{b}^1) && \text{(Definition 6)}
\end{aligned}$$

□

We extend the definitions of bin and proj to vectors of ring elements in the natural sense. That is, let $\boldsymbol{a} \in \mathcal{R}_q^{\xi}$ and $\boldsymbol{b} \in \mathcal{R}_q^{\xi \cdot \lceil \log q \rceil}$ with $\boldsymbol{b}_i := (b_{i \cdot \lceil \log q \rceil}, \ldots, b_{(i+1) \cdot \lceil \log q \rceil - 1})^\intercal$, then $\mathsf{bin}(\boldsymbol{a}) := (\mathsf{bin}(a_0), \ldots, \mathsf{bin}(a_\xi))$ and $\mathsf{proj}(\boldsymbol{b}) := (\mathsf{proj}(\boldsymbol{b}_0), \ldots, \mathsf{proj}(\boldsymbol{b}_{\xi-1}))$. It is easy to verify that Lemma 3 and Lemma 4 also apply to this extension.

Equipped with the decomposition and projection functions, we are now ready to define how the labels of the nodes in our tree construction will be computed.

Definition 7 (Labelled full binary tree). *Let* $\boldsymbol{m} = (m_0, \ldots, m_{2^\tau - 1})^\intercal \in \mathcal{R}_q^{2^\tau}$ *and* $\boldsymbol{h}_0, \boldsymbol{h}_1 \in \mathcal{R}_q^{\lceil \log q \rceil}$ *be fixed. We define the labeling function* $\mathsf{label} : \{0,1\}^{\leq \tau} \to \mathcal{R}_q^{\lceil \log q \rceil}$ *of for a labeled full binary tree of depth* τ *as*

$$\mathsf{label}(\boldsymbol{h}_0, \boldsymbol{h}_1, \boldsymbol{m}, v) := \begin{cases} \mathsf{bin}(m_v) & \text{if } |v| = \tau \\ \mathsf{bin}\begin{pmatrix} \boldsymbol{h}_0^\intercal \cdot \mathsf{label}(\boldsymbol{h}_0, \boldsymbol{h}_1, \boldsymbol{m}, v\|0) \\ + \boldsymbol{h}_1^\intercal \cdot \mathsf{label}(\boldsymbol{h}_0, \boldsymbol{h}_1, \boldsymbol{m}, v\|1) \end{pmatrix} & \text{if } |v| < \tau \end{cases}$$

$\mathsf{Setup}(1^\lambda, \tau)$

$\boldsymbol{h}_0 \leftarrow \mathcal{R}_q^{\lceil \log q \rceil}$

$\boldsymbol{h}_1 \leftarrow \mathcal{R}_q^{\lceil \log q \rceil}$

return $(\boldsymbol{h}_0, \boldsymbol{h}_1)$

$\mathsf{Com}(\mathsf{pp}, \boldsymbol{m})$

$\boldsymbol{c} := \mathsf{label}(\boldsymbol{h}_0, \boldsymbol{h}_1, \boldsymbol{m}, \epsilon)$

return $\boldsymbol{c}$

$\mathsf{Open}(\mathsf{pp}, \boldsymbol{c}, \boldsymbol{m}, t)$

$\tilde{t} := \mathsf{bin}_{\mathbb{N}}(t)$

for $j \in [\tau - 2]$

$\quad \boldsymbol{p}_{j+1} := \mathsf{label}(\boldsymbol{h}_0, \boldsymbol{h}_1, \boldsymbol{m}, \tilde{t}_{<j} \| \tilde{t}_j)$

$\quad \boldsymbol{s}_{j+1} := \mathsf{label}(\boldsymbol{h}_0, \boldsymbol{h}_1, \boldsymbol{m}, \tilde{t}_{<j} \| (\tilde{t}_j \oplus 1))$

return $(\boldsymbol{p}_1, \ldots, \boldsymbol{p}_\tau, \boldsymbol{s}_1, \ldots, \boldsymbol{s}_\tau)$

$\mathsf{wVrfy}(\mathsf{pp}, \boldsymbol{c}, t, \boldsymbol{d})$

return $\mathsf{Vrfy}(\mathsf{pp}, \boldsymbol{c}, t, \boldsymbol{d}, 2\rho\alpha)$

$\mathsf{sVrfy}(\mathsf{pp}, \boldsymbol{c}, t, \boldsymbol{d})$

return $\mathsf{Vrfy}(\mathsf{pp}, \boldsymbol{c}, t, \boldsymbol{d}, \rho\alpha)$

$\mathsf{Vrfy}(\mathsf{pp}, \boldsymbol{c}, t, \boldsymbol{d}, \beta')$

parse $\boldsymbol{d}$ **as** $(\boldsymbol{p}_1, \ldots, \boldsymbol{p}_\tau, \boldsymbol{s}_1, \ldots, \boldsymbol{s}_\tau)$

$\tilde{t} := \mathsf{bin}_{\mathbb{N}}(t)$

$\boldsymbol{p}_0 := \boldsymbol{c}$

for $j \in [\tau - 1]$

$\quad$ **if** $\|\boldsymbol{p}_{j+1}\| > \beta'$ **or** $\|\boldsymbol{s}_{j+1}\| > \beta'$

$\quad\quad$ **return** $\perp$

$\quad$ **if** $\mathsf{proj}(\boldsymbol{p}_j) \neq \boldsymbol{h}_{\tilde{t}_j}^\intercal \cdot \boldsymbol{p}_{j+1} + \boldsymbol{h}_{\tilde{t}_j \oplus 1}^\intercal \cdot \boldsymbol{s}_{j+1}$

$\quad\quad$ **return** $\perp$

return $\mathsf{proj}(\boldsymbol{p}_\tau)$

Fig. 1. The construction of a homomorphic vector commitment for $\mathcal{R}_q$ based on a labeled binary tree.

Our construction proceeds by effectively computing a Merkle tree on top of a given input vector, where the labels of the nodes are computed as specified in Definition 7. The root node of that tree will constitute the vector commitment. To open a specific position in the vector, we will output all the node labels and adjacent node labels along the path from that position in the vector to the root of the computed tree.

Theorem 5. *Let $\mathcal{R}_q = \mathbb{Z}_q[x]/\langle x^n + 1\rangle$ be a polynomial ring parameterized by $n = \mathsf{poly}(\lambda)$ and $q = \mathsf{poly}(\lambda)$. Let α be the smallest integer, such that $\binom{n}{\alpha} \cdot 2^\alpha \geq 2^\lambda$. If the $\mathsf{SIS}_{\mathcal{R},q,2\lceil \log q \rceil, 2\rho\alpha}$ problem is hard, then the construction from Figure 1 is a $(\rho, \mathcal{T}_\alpha)$-homomorphically correct, robustly homomorphic, and position binding vector commitment scheme (HVC) for $\mathcal{R}_q$.*

Proof. The theorem follows from Lemma 6, Lemma 8, and Lemma 9 proven below. □

Lemma 6. *The construction from Figure 1 is a $(\rho, \mathcal{T}_\alpha)$-homomorphically correct vector commitment scheme (HVC) for $\mathcal{R}_q$.*

Proof. Let $\boldsymbol{m}^0, \ldots, \boldsymbol{m}^{\ell-1} \in \mathcal{R}_q^{2^\tau}$, $\boldsymbol{p}_0^i = \mathsf{Com}(\mathsf{pp}, \boldsymbol{m}^i)$, $t \in [2^\tau - 1]$, $(\boldsymbol{p}_1^i, \ldots, \boldsymbol{p}_\tau^i, \boldsymbol{s}_1^i, \ldots, \boldsymbol{s}_\tau^i)^\intercal = \mathsf{Open}(\mathsf{pp}, \boldsymbol{p}_0^i, \boldsymbol{m}^i, t)$, and $w^0, \ldots, w^{\ell-1} \in \mathcal{T}_\alpha$ as specified in Definition 2. We will first prove a claim about the individual honestly computed commitments and decommitments.

Claim 7. *For all $j \in [\tau - 1]$ it holds that $\mathsf{proj}(\boldsymbol{p}_j^i) = \boldsymbol{h}_{\tilde{t}_j}^\intercal \cdot \boldsymbol{p}_{j+1}^i + \boldsymbol{h}_{1-\tilde{t}_j}^\intercal \cdot \boldsymbol{s}_{j+1}^i$.*

Proof. We observe that for all $j \in [\tau - 1]$ it holds that

$$\mathsf{proj}(\boldsymbol{p}_j^i)$$

$$
\begin{aligned}
&= \mathsf{proj}\left(\mathsf{label}(\boldsymbol{h}_0, \boldsymbol{h}_1, \boldsymbol{m}^i, \tilde{t}_{<j})\right) && \text{(Def. of Com and Open)}\\
&= \mathsf{proj}\left(\mathsf{bin}\begin{pmatrix} \boldsymbol{h}_0^\intercal \cdot \mathsf{label}(\boldsymbol{h}_0, \boldsymbol{h}_1, \boldsymbol{m}^i, \tilde{t}_{<j} \| 0) \\ +\boldsymbol{h}_1^\intercal \cdot \mathsf{label}(\boldsymbol{h}_0, \boldsymbol{h}_1, \boldsymbol{m}^i, \tilde{t}_{<j} \| 1) \end{pmatrix}\right) && \text{(Definition 7)}\\
&= \boldsymbol{h}_0^\intercal \cdot \mathsf{label}(\boldsymbol{h}_0, \boldsymbol{h}_1, \boldsymbol{m}^i, \tilde{t}_{<j} \| 0) + \boldsymbol{h}_1^\intercal \cdot \mathsf{label}(\boldsymbol{h}_0, \boldsymbol{h}_1, \boldsymbol{m}^i, \tilde{t}_{<j} \| 1) && \text{(Lemma 3)}\\
&= \boldsymbol{h}_{\tilde{t}_j}^\intercal \cdot \mathsf{label}(\boldsymbol{h}_0, \boldsymbol{h}_1, \boldsymbol{m}^i, \tilde{t}_{<j} \| \tilde{t}_j) + \boldsymbol{h}_{\tilde{t}_j \oplus 1}^\intercal \cdot \mathsf{label}(\boldsymbol{h}_0, \boldsymbol{h}_1, \boldsymbol{m}^i, \tilde{t}_{<j} \| (\tilde{t}_j \oplus 1))\\
&= \boldsymbol{h}_{\tilde{t}_j}^\intercal \cdot \boldsymbol{p}_{j+1}^i + \boldsymbol{h}_{\tilde{t}_j \oplus 1}^\intercal \cdot \boldsymbol{s}_{j+1}^i && \text{(Def. of Open)}
\end{aligned}
$$

as claimed. □

We are now ready to prove Lemma 6. We first note that for all $j \in [\tau]$ it holds that

$$
\left\| \sum_{i=0}^{\ell-1} w^i \cdot \boldsymbol{p}_j^i \right\| \leq \ell \cdot \max_{i \in [\ell-1]} \left\{ \left\| w^i \cdot \boldsymbol{p}_j^i \right\| \right\} \overset{\text{Lemma 1}}{\leq} \ell \cdot \alpha \leq \rho\alpha
$$

$$
\left\| \sum_{i=0}^{\ell-1} w^i \cdot \boldsymbol{s}_j^i \right\| \leq \ell \cdot \max_{i \in [\ell-1]} \left\{ \left\| w^i \cdot \boldsymbol{s}_j^i \right\| \right\} \overset{\text{Lemma 1}}{\leq} \ell \cdot \alpha \leq \rho\alpha.
$$

Further, for all $j \in [\tau - 1]$ it holds that

$$
\begin{aligned}
\mathsf{proj}\Big(\sum_{i=0}^{\ell-1} w^i \cdot \boldsymbol{p}_j^i\Big) &= \sum_{i=0}^{\ell-1} w^i \cdot \mathsf{proj}(\boldsymbol{p}_j^i) && \text{(Lemma 4)}\\
&= \sum_{i=0}^{\ell-1} w^i \cdot (\boldsymbol{h}_{\tilde{t}_j}^\intercal \cdot \boldsymbol{p}_{j+1}^i + \boldsymbol{h}_{1-\tilde{t}_j}^\intercal \cdot \boldsymbol{s}_{j+1}^i) && \text{(Claim 7)}\\
&= \sum_{i=0}^{\ell-1} \boldsymbol{h}_{\tilde{t}_j}^\intercal \cdot w^i \cdot \boldsymbol{p}_{j+1}^i + \boldsymbol{h}_{1-\tilde{t}_j}^\intercal \cdot w^i \cdot \boldsymbol{s}_{j+1}^i\\
&= \boldsymbol{h}_{\tilde{t}_j}^\intercal \cdot \Big(\sum_{i=0}^{\ell-1} w^i \cdot \boldsymbol{p}_{j+1}^i\Big) + \boldsymbol{h}_{1-\tilde{t}_j}^\intercal \cdot \Big(\sum_{i=0}^{\ell-1} w^i \cdot \boldsymbol{s}_{j+1}^i\Big)
\end{aligned}
$$

Therefore, all checks in the strong verification algorithm will go through and it will output

$$
\begin{aligned}
\mathsf{proj}\Big(\sum_{i=0}^{\ell-1} w^i \cdot \boldsymbol{p}_\tau^i\Big) &= \sum_{i=0}^{\ell-1} w^i \cdot \mathsf{proj}(\boldsymbol{p}_\tau^i) && \text{(Lemma 4)}\\
&= \sum_{i=0}^{\ell-1} w^i \cdot \mathsf{proj}(\mathsf{bin}(m_t^i)) && \text{(Def. of Open)}\\
&= \sum_{i=0}^{\ell-1} w^i \cdot m_t^i && \text{(Lemma 3)}
\end{aligned}
$$

as required by Definition 2. □

Lemma 8. *The construction from Figure 1 is a robustly homomorphic vector commitment scheme.*

Proof. Let $\boldsymbol{c}^0, \boldsymbol{c}^1 \in \mathcal{R}_q^{\ell_{\mathsf{com}}}$, and $\boldsymbol{d}^0, \boldsymbol{d}^1 \in \mathcal{R}_q^{\ell_{\mathsf{dec}}}$, and $t \in [2^\tau - 1]$ be arbitrary, such that

$$\mathsf{sVrfy}(\mathsf{pp}, \boldsymbol{c}^0, t, \boldsymbol{d}^0) = m^0 \quad \text{and} \quad \mathsf{sVrfy}(\mathsf{pp}, \boldsymbol{c}^1, t, \boldsymbol{d}^1) = m^1 \tag{1}$$

with $m^0, m^1 \neq \bot$. Let $\boldsymbol{d}^i$ parse as $(\boldsymbol{p}_1^i, \ldots, \boldsymbol{p}_\tau^i, \boldsymbol{s}_1^i, \ldots, \boldsymbol{s}_\tau^i)^\intercal$ for $i \in \{0,1\}$. We first note that *if* $\mathsf{wVrfy}(\mathsf{pp}, \boldsymbol{c}^0 - \boldsymbol{c}^1, t, \boldsymbol{d}^0 - \boldsymbol{d}^1) \neq \bot$, then

$$\begin{aligned}
&\mathsf{wVrfy}(\mathsf{pp}, \boldsymbol{c}^0 - \boldsymbol{c}^1, t, \boldsymbol{d}^0 - \boldsymbol{d}^1) \\
=&\mathsf{proj}(\boldsymbol{p}_\tau^0 - \boldsymbol{p}_\tau^1) && \text{(Def of Vrfy)} \\
=&\mathsf{proj}(\boldsymbol{p}_\tau^0) - \mathsf{proj}(\boldsymbol{p}_\tau^1) && \text{(Lemma 4)} \\
=&\mathsf{sVrfy}(\mathsf{pp}, \boldsymbol{c}^0, t, \boldsymbol{d}^0) - \mathsf{sVrfy}(\mathsf{pp}, \boldsymbol{c}^1, t, \boldsymbol{d}^1) && \text{(Def. of sVrfy)} \\
=&m^0 - m^1. && \text{(Equation 1)}
\end{aligned}$$

It thus remains to show that $\mathsf{wVrfy}(\mathsf{pp}, \boldsymbol{c}^0 - \boldsymbol{c}^1, t, \boldsymbol{d}^0 - \boldsymbol{d}^1) \neq \bot$. For this, let further $\boldsymbol{p}_0^i = \boldsymbol{c}^i$. By definition of the strong verification algorithm, and since $m^0, m^1 \neq \bot$ it holds that for $i \in \{0,1\}$ and $j \in [\tau - 1]$

$$\left\|\boldsymbol{p}_{j+1}^i\right\| \leq \rho\alpha \qquad \left\|\boldsymbol{s}_{j+1}^i\right\| \leq \rho\alpha \tag{2}$$

$$\mathsf{proj}(\boldsymbol{p}_j^i) = \boldsymbol{h}_{\tilde{t}_j}^\intercal \cdot \boldsymbol{p}_{j+1}^i + \boldsymbol{h}_{1-\tilde{t}_j}^\intercal \cdot \boldsymbol{s}_{j+1}^i. \tag{3}$$

From Equation 2 it follows that for all $j \in [\tau - 1]$

$$\begin{aligned}
\left\|\boldsymbol{p}_j^0 - \boldsymbol{p}_j^1\right\| \leq& \left\|\boldsymbol{p}_j^0\right\| + \left\|\boldsymbol{p}_j^1\right\| \leq 2\rho\alpha \\
\left\|\boldsymbol{s}_j^0 - \boldsymbol{s}_j^1\right\| \leq& \left\|\boldsymbol{s}_j^0\right\| + \left\|\boldsymbol{s}_j^1\right\| \leq 2\rho\alpha.
\end{aligned}$$

From Equation 3 and the linearity of proj it follows that for all $j \in [\tau - 1]$

$$\begin{aligned}
&\mathsf{proj}(\boldsymbol{p}_j^0 - \boldsymbol{p}_j^1) \\
=&\mathsf{proj}(\boldsymbol{p}_j^0) - \mathsf{proj}(\boldsymbol{p}_j^1) && \text{(Lemma 4)} \\
=&(\boldsymbol{h}_{\tilde{t}_j}^\intercal \cdot \boldsymbol{p}_{j+1}^0 + \boldsymbol{h}_{1-\tilde{t}_j}^\intercal \cdot \boldsymbol{s}_{j+1}^0) - (\boldsymbol{h}_{\tilde{t}_j}^\intercal \cdot \boldsymbol{p}_{j+1}^1 + \boldsymbol{h}_{1-\tilde{t}_j}^\intercal \cdot \boldsymbol{s}_{j+1}^1) && \text{(Equation 3)} \\
=&\boldsymbol{h}_{\tilde{t}_j}^\intercal \cdot (\boldsymbol{p}_{j+1}^0 - \boldsymbol{p}_{j+1}^1) + \boldsymbol{h}_{1-\tilde{t}_j}^\intercal \cdot (\boldsymbol{s}_{j+1}^0 - \boldsymbol{s}_{j+1}^1).
\end{aligned}$$

Therefore, all checks in the weak verification algorithm go through and $\mathsf{wVrfy}(\mathsf{pp}, \boldsymbol{c}^0 - \boldsymbol{c}^1, t, \boldsymbol{d}^0 - \boldsymbol{d}^1) \neq \bot$. □

Lemma 9. *If the* $\mathsf{SIS}_{\mathcal{R},q,2\lceil \log q \rceil, 4\rho\alpha}$ *problem is hard then the construction from Figure 1 is position binding.*

Proof. We will prove this lemma by leveraging that any pair of valid decommitments will lead to a collision somewhere in the generalized hash tree, which can be turned into a solution for the SIS instance. Let $\mathcal{A}$ be an arbitrary PPT adversary against the position binding property of the construction. We construct a PPT algorithm that solves the $\mathsf{SIS}_{\mathcal{R},q,2\lceil \log q \rceil, 4\rho\alpha}$ problem as follows. Upon input $\boldsymbol{a} = (a_0, \ldots, a_{2\lceil \log q \rceil - 1})^\intercal$, $\mathcal{B}$ sets $\boldsymbol{h}_0 := (a_0, \ldots, a_{\lceil \log q \rceil - 1})^\intercal$ and $\boldsymbol{h}_1 := (a_{\lceil \log q \rceil}, \ldots, a_{2\lceil \log q \rceil - 1})^\intercal$ and runs $(\boldsymbol{c}, t, \boldsymbol{d}^0, \boldsymbol{d}^1) \leftarrow \mathcal{A}((\boldsymbol{h}_0, \boldsymbol{h}_1))$. For $i \in \{0,1\}$ let $m^i := \mathsf{wVrfy}((\boldsymbol{h}_0, \boldsymbol{h}_1), \boldsymbol{c}, t, \boldsymbol{d}^i)$. If $m^0 = m^1$ or $\bot \in \{m^0, m^1\}$, $\mathcal{B}$ aborts. Otherwise, parse $\boldsymbol{d}^i$ as $(\boldsymbol{p}_1^i, \ldots, \boldsymbol{p}_\tau^i, \boldsymbol{s}_1^i, \ldots, \boldsymbol{s}_\tau^i)$, set $\boldsymbol{p}_0^i := \boldsymbol{c}$.

Let $j^* \in [\tau]$ be the *largest* index, such that $\mathsf{proj}(\boldsymbol{p}^0_{j^*}) = \mathsf{proj}(\boldsymbol{p}^1_{j^*})$. Note that such an index always exists, since $\boldsymbol{p}^0_0 = \boldsymbol{c} = \boldsymbol{p}^1_0$ and that $j^* < \tau$, since $\mathsf{proj}(\boldsymbol{p}^0_{j^*}) = m^0 \neq m^1 = \mathsf{proj}(\boldsymbol{p}^1_{j^*})$. If $\tilde{t}_{j^*} = 0$, $\mathcal{B}$ outputs $\boldsymbol{z} := (\boldsymbol{p}^0_{j^*+1}, \boldsymbol{s}^0_{j^*+1})^\intercal - (\boldsymbol{p}^1_{j^*+1}, \boldsymbol{s}^1_{j^*+1})^\intercal$, if $\tilde{t}_{j^*} = 1$, $\mathcal{B}$ outputs $\boldsymbol{z} := (\boldsymbol{s}^0_{j^*+1}, \boldsymbol{p}^0_{j^*+1})^\intercal - (\boldsymbol{s}^1_{j^*+1}, \boldsymbol{p}^1_{j^*+1})^\intercal$.

We now analyze the success probability of $\mathcal{B}$. It holds that $\mathsf{proj}(\boldsymbol{p}^0_{j^*}) = \mathsf{proj}(\boldsymbol{p}^1_{j^*})$ and by the definition of the weak verification algorithm that

$$\begin{aligned}
&\boldsymbol{h}^\intercal_{\tilde{t}_{j^*}} \cdot \boldsymbol{p}^0_{j^*+1} + \boldsymbol{h}^\intercal_{\tilde{t}_{j^*}\oplus 1} \cdot \boldsymbol{s}^0_{j^*+1} = \boldsymbol{h}^\intercal_{\tilde{t}_{j^*}} \cdot \boldsymbol{p}^1_{j^*+1} + \boldsymbol{h}^\intercal_{\tilde{t}_{j^*}\oplus 1} \cdot \boldsymbol{s}^1_{j^*+1} \\
\iff &\boldsymbol{h}^\intercal_{\tilde{t}_{j^*}} \cdot (\boldsymbol{p}^0_{j^*+1} - \boldsymbol{p}^1_{j^*+1}) + \boldsymbol{h}^\intercal_{\tilde{t}_{j^*}\oplus 1} \cdot (\boldsymbol{s}^0_{j^*+1} - \boldsymbol{s}^1_{j^*+1}) = 0 \\
\iff &\boldsymbol{a}^\intercal \cdot \boldsymbol{z} = \boldsymbol{0}
\end{aligned}$$

It further holds by the definition of the weak verification algorithm that

$$\left\|\boldsymbol{p}^0_{j^*+1}\right\| \leq 2\rho\alpha, \quad \left\|\boldsymbol{s}^0_{j^*+1}\right\| \leq 2\rho\alpha, \quad \left\|\boldsymbol{p}^1_{j^*+1}\right\| \leq 2\rho\alpha, \quad \left\|\boldsymbol{s}^1_{j^*+1}\right\| \leq 2\rho\alpha.$$

Therefore, the norm of $\boldsymbol{z}$ can be bounded as

$$\|\boldsymbol{z}\| \leq \max\{\left\|\boldsymbol{p}^0_{j^*+1}\right\|, \left\|\boldsymbol{s}^0_{j^*+1}\right\|\} + \max\{\left\|\boldsymbol{p}^1_{j^*+1}\right\|, \left\|\boldsymbol{s}^1_{j^*+1}\right\|\} \leq 4\rho\alpha.$$

It remains to show that $\boldsymbol{z} \neq 0$. Since j^* is the *largest* index such that

$$\mathsf{proj}(\boldsymbol{p}^0_{j^*}) = \mathsf{proj}(\boldsymbol{p}^1_{j^*})$$

it holds that

$$\mathsf{proj}(\boldsymbol{p}^0_{j^*+1}) \neq \mathsf{proj}(\boldsymbol{p}^1_{j^*+1})$$

and thereby that

$$\boldsymbol{p}^0_{j^*+1} \neq \boldsymbol{p}^1_{j^*+1}.$$

Therefore $\boldsymbol{z} \neq \boldsymbol{0}$. Thus, whenever $\mathcal{A}$ is successful, $\mathcal{B}$ is successful with probability 1 and we can conclude that

$$\begin{aligned}
\mathsf{negl}(\lambda) &\geq \Pr[\boldsymbol{a} \leftarrow \mathcal{R}_q^{2\lceil \log q \rceil}; \boldsymbol{s} \leftarrow \mathcal{B}(\boldsymbol{a}) : \boldsymbol{z} \in \mathcal{B}_{4\rho\alpha}^{2\lceil \log q \rceil} \setminus \{\boldsymbol{0}\} \wedge \boldsymbol{a}^\intercal \boldsymbol{s} = 0] \\
&= \Pr\left[\begin{array}{rl} \mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda, \tau); & \\ (c, t, d_1, d_2) \leftarrow \mathcal{A}(\mathsf{pp}); & \\ m_1 \leftarrow \mathsf{wVrfy}(\mathsf{pp}, c, t, d_1); & : \begin{array}{r} m_1 \neq m_2 \\ \wedge \perp \notin \{m_1, m_2\} \end{array} \\ m_2 \leftarrow \mathsf{wVrfy}(\mathsf{pp}, c, t, d_2) & \end{array}\right]
\end{aligned}$$

□

3.2 Homomorphic Vector Commitment for $\mathcal{R}_q^\xi$

In the previous section we constructed an HVC for domain $\mathcal{R}_{\bar{q}}$ for some $\bar{q} = \mathsf{poly}(\lambda)$. For our application however, this is however not ideal for two reasons. In our main construction of a synchronized multi-signature scheme, the committed values are public keys of a one-time signature scheme. These are not individual ring elements, but pairs of $\mathcal{R}_q$ elements for some $q = \mathsf{poly}(\lambda)$ leading to a domain mismatch. The simplest solution of choosing $\bar{q} = q$ and always decommitting to pairs of leaves works but turns out to be inefficient. We therefore want the freedom to choose

$\mathsf{Setup}(1^\lambda, \tau)$

$\overline{\mathsf{pp}} \leftarrow \overline{\mathsf{Setup}}(1^\lambda, \tau)$
$\boldsymbol{h} \leftarrow \mathcal{R}_{\bar{q}}^{\ell \cdot \lceil \log q \rceil}$
return $(\mathsf{pp}', \boldsymbol{h})$

$\mathsf{Com}(\mathsf{pp}, \boldsymbol{m})$

$\bar{\boldsymbol{m}} := (\boldsymbol{h}^\intercal \cdot \mathsf{bin}_q(\boldsymbol{m}_0), \ldots, \boldsymbol{h}^\intercal \cdot \mathsf{bin}_q(\boldsymbol{m}_{\tau-1}))^\intercal$
return $\overline{\mathsf{Com}}(\overline{\mathsf{pp}}, \bar{\boldsymbol{m}})$

$\mathsf{Open}(\mathsf{pp}, \boldsymbol{c}, \boldsymbol{m}, t)$

$\bar{\boldsymbol{m}} := (\boldsymbol{h}^\intercal \cdot \mathsf{bin}_q(\boldsymbol{m}_0), \ldots, \boldsymbol{h}^\intercal \cdot \mathsf{bin}_q(\boldsymbol{m}_{\tau-1}))^\intercal$
return $(\overline{\mathsf{Open}}(\overline{\mathsf{pp}}, \boldsymbol{c}, \bar{\boldsymbol{m}}, t), \mathsf{bin}(\boldsymbol{m}_t))$

$\mathsf{wVrfy}(\mathsf{pp}, \boldsymbol{c}, t, (\bar{\boldsymbol{d}}, \boldsymbol{b}))$

if $\|\boldsymbol{b}\| > 2\rho\alpha$
  return $\perp$
if $\overline{\mathsf{wVrfy}}(\overline{\mathsf{pp}}, \boldsymbol{c}, t, \bar{\boldsymbol{d}}) \neq \boldsymbol{h}^\intercal \cdot \boldsymbol{b}$
  return $\perp$
return $\mathsf{proj}_q(\boldsymbol{b})$

$\mathsf{sVrfy}(\mathsf{pp}, \boldsymbol{c}, t, (\bar{\boldsymbol{d}}, \boldsymbol{b}))$

if $\|\boldsymbol{b}\| > \rho\alpha$
  return $\perp$
if $\overline{\mathsf{sVrfy}}(\overline{\mathsf{pp}}, \boldsymbol{c}, t, \bar{\boldsymbol{d}}) \neq \boldsymbol{h}^\intercal \cdot \boldsymbol{b}$
  return $\perp$
return $\mathsf{proj}_q(\boldsymbol{b})$

Fig. 2. The construction of a homomorphic vector commitment for $\mathcal{R}_q^\xi$ based on a homomorphic vector commitment for $\mathcal{R}_{\bar{q}}$.

$\bar{q} \neq q$. For this purpose we describe a domain extension in the following, that allows us to leverage the HVC for domain $\mathcal{R}_{\bar{q}}$ into an HVC for domain $\mathcal{R}_q^\xi$.

Given a vector commitment with domain X it is very simple to construct a vector commitment for an arbitrary domain Y, simply by applying a collision resistant hash function $H : Y \to X$ to the committed elements. In our case we need to take care to choose the hash function in such a way to maintain the homomorphism. This is easily done by again applying Ajtai's hash function combined with binary decomposition.

Theorem 10. *Let $\mathcal{R}_q = \mathbb{Z}_q[x]/\langle x^n+1\rangle$ and $\mathcal{R}_{\bar{q}} = \mathbb{Z}_{\bar{q}}[x]/\langle x^n+1\rangle$ be polynomial rings parameterized by $n = \mathsf{poly}(\lambda)$ and $q = \mathsf{poly}(\lambda)$, $\bar{q} = \mathsf{poly}(\lambda)$ respectively and let $\xi \in \mathbb{N}$. If $\overline{\mathsf{HVC}}$ is a $(\rho, \mathcal{T}_\alpha)$-homomorphically correct, robustly homomorphic, and position binding vector commitment scheme (HVC) for $\mathcal{R}_{\bar{q}}$ and if the $\mathsf{SIS}_{\mathcal{R},\bar{q},\xi\lceil\log q\rceil,4\rho\alpha}$ problem is hard, then the construction from Figure 2 is a $(\rho, \mathcal{T}_\alpha)$-homomorphically correct, robustly homomorphic, and position binding vector commitment scheme (HVC) for $\mathcal{R}_q^\xi$.*

Proof. The theorem follows from Lemma 11, Lemma 12, and Lemma 13 proven below. □

Lemma 11. *If $\overline{\mathsf{HVC}}$ is a $(\rho, \mathcal{T}_\alpha)$-homomorphically correct vector commitment scheme (HVC) for $\mathcal{R}_{\bar{q}}$, then the construction from Figure 2 is a $(\rho, \mathcal{T}_\alpha)$-homomorphically correct vector commitment scheme (HVC) for $\mathcal{R}_q^\xi$.*

Proof. Let $\boldsymbol{m}^0, \ldots, \boldsymbol{m}^{\ell-1} \in \left(\mathcal{R}_q^\xi\right)^{2^\tau}$, $\boldsymbol{c}^i = \mathsf{Com}(\mathsf{pp}, \boldsymbol{m}^i)$, $t \in [2^\tau - 1]$, $(\bar{\boldsymbol{d}}^i, \boldsymbol{b}^i)^\intercal = \mathsf{Open}(\mathsf{pp}, \boldsymbol{c}^i, \boldsymbol{m}^i, t)$, and $w^0, \ldots, w^{\ell-1} \in \mathcal{T}_\alpha$ as specified in Definition 2.

By the $(\rho, \mathcal{T}_\alpha)$-homomorphic correctness of $\overline{\mathsf{HVC}}$ and the definition of $\boldsymbol{b}^i$, it holds that

$$\overline{\mathsf{sVrfy}}(\overline{\mathsf{pp}}, \sum_{i=0}^{\ell-1} w^i \cdot \boldsymbol{c}^i, t, \sum_{i=0}^{\ell-1} w^i \cdot \bar{\boldsymbol{d}}^i) = \sum_{i=0}^{\ell-1} w^i \cdot (\boldsymbol{h}^\intercal \cdot \mathsf{bin}(\boldsymbol{m}_t^i)) = \boldsymbol{h}^\intercal \cdot \sum_{i=0}^{\ell-1} w^i \cdot \boldsymbol{b}^i$$

Further, since all $\boldsymbol{b}^i$ are binary and $w^i \in \mathcal{T}_\alpha$, it holds that

$$\left\| \sum_{i=0}^{\ell-1} w^i \cdot \boldsymbol{b}_t^i \right\| \overset{\text{Lemma 1}}{\leq} \ell \cdot \alpha \leq \rho\alpha.$$

Therefore, all checks in the strong verification algorithm go through and it outputs

$$\begin{aligned}
\mathsf{proj}(\sum_{i=0}^{\ell-1} w^i \cdot \boldsymbol{b}^i) &= \sum_{i=0}^{\ell-1} w^i \cdot \mathsf{proj}(\boldsymbol{b}^i) \\
&= \sum_{i=0}^{\ell-1} w^i \cdot \mathsf{proj}(\mathsf{bin}(\boldsymbol{m}_t^i)) \\
&= \sum_{i=0}^{\ell-1} w^i \cdot \boldsymbol{m}_t^i
\end{aligned}$$

as required. □

Lemma 12. *If* $\overline{\mathsf{HVC}}$ *is robustly homomorphic, then the construction from Figure 2 is a robustly homomorphic vector commitment scheme (HVC).*

Proof. Let $\boldsymbol{c}^0, \boldsymbol{c}^1 \in \mathcal{R}_q^{\ell_{\mathsf{com}}}$, $(\bar{\boldsymbol{d}}^0, \boldsymbol{b}^0), (\bar{\boldsymbol{d}}^1, \boldsymbol{b}^1) \in \mathcal{R}_q^{\ell_{\mathsf{dec}}}$, and $t \in [2^\tau - 1]$ be arbitrary, such that

$$\mathsf{sVrfy}(\mathsf{pp}, \boldsymbol{c}^0, t, (\bar{\boldsymbol{d}}^0, \boldsymbol{b}^0)) = \boldsymbol{m}^0 \quad \text{and} \quad \mathsf{sVrfy}(\mathsf{pp}, \boldsymbol{c}^1, t, (\bar{\boldsymbol{d}}^1, \boldsymbol{b}^1)) = \boldsymbol{m}^1 \tag{4}$$

with $\boldsymbol{m}^0, \boldsymbol{m}^1 \neq \perp$. We first note that *if* $\mathsf{wVrfy}(\mathsf{pp}, \boldsymbol{c}^0 - \boldsymbol{c}^1, t, (\bar{\boldsymbol{d}}^0 - \bar{\boldsymbol{d}}^1, \boldsymbol{b}^0 - \boldsymbol{b}^1)) \neq \perp$, then

$$\begin{aligned}
&\mathsf{wVrfy}(\mathsf{pp}, \boldsymbol{c}^0 - \boldsymbol{c}^1, t, (\bar{\boldsymbol{d}}^0 - \bar{\boldsymbol{d}}^1, \boldsymbol{b}^0 - \boldsymbol{b}^1)) \\
=&\mathsf{proj}(\boldsymbol{b}^0 - \boldsymbol{b}^1) && \text{(Def of } \mathsf{wVrfy}) \\
=&\mathsf{proj}(\boldsymbol{b}^0) - \mathsf{proj}(\boldsymbol{b}^1) && \text{(Lemma 4)} \\
=&\mathsf{sVrfy}(\mathsf{pp}, \boldsymbol{c}^0, t, (\bar{\boldsymbol{d}}^0, \boldsymbol{b}^0)) - \mathsf{sVrfy}(\mathsf{pp}, \boldsymbol{c}^1, t, (\bar{\boldsymbol{d}}^1, \boldsymbol{b}^1)) && \text{(Def. of } \mathsf{sVrfy}) \\
=&\boldsymbol{m}^0 - \boldsymbol{m}^1. && \text{(Equation 4)}
\end{aligned}$$

It thus remains to show that $\mathsf{wVrfy}(\mathsf{pp}, \boldsymbol{c}^0 - \boldsymbol{c}^1, t, (\bar{\boldsymbol{d}}^0 - \bar{\boldsymbol{d}}^1, \boldsymbol{b}^0 - \boldsymbol{b}^1)) \neq \perp$.

We first note that the norm check goes through, since $\left\|\boldsymbol{b}^0 - \boldsymbol{b}^0\right\| \leq \left\|\boldsymbol{b}^0\right\| + \left\|\boldsymbol{b}^1\right\| \leq 2\rho\alpha$, where the last inequality follows from the definition of the strong verification algorithm and the fact that $\boldsymbol{m}^0, \boldsymbol{m}^1 \neq \perp$. From the same observation it also follows that

$$\overline{\mathsf{sVrfy}}(\overline{\mathsf{pp}}, \boldsymbol{c}^0, t, \bar{\boldsymbol{d}}^0) = \boldsymbol{h}^\intercal \cdot \boldsymbol{b}^0 \quad \text{and} \quad \overline{\mathsf{sVrfy}}(\overline{\mathsf{pp}}, \boldsymbol{c}^1, t, \bar{\boldsymbol{d}}^1) = \boldsymbol{h}^\intercal \cdot \boldsymbol{b}^1.$$

Therefore, since $\overline{\mathsf{HVC}}$ is robustly homomorphic it follows that

$$\overline{\mathsf{wVrfy}}(\overline{\mathsf{pp}}, \boldsymbol{c}^0 - \boldsymbol{c}^1, t, \bar{\boldsymbol{d}}^0 - \bar{\boldsymbol{d}}^1) = \boldsymbol{h}^\intercal \cdot \boldsymbol{b}^0 - \boldsymbol{h}^\intercal \cdot \boldsymbol{b}^1 = \boldsymbol{h}^\intercal \cdot (\boldsymbol{b}^0 - \boldsymbol{b}^1).$$

Since all checks go through, it follows that $\mathsf{wVrfy}(\mathsf{pp}, \boldsymbol{c}^0 - \boldsymbol{c}^1, t, (\bar{\boldsymbol{d}}^0 - \bar{\boldsymbol{d}}^1, \boldsymbol{b}^0 - \boldsymbol{b}^1)) \neq \perp$. □

Lemma 13. *If* $\overline{\mathsf{HVC}}$ *is position binding and if the* $\mathsf{SIS}_{\mathcal{R},q,\xi\lceil \log q \rceil, 4\rho\alpha}$ *problem is hard then the construction from Figure 2 is position binding.*

Proof. Let $\mathcal{A}$ be an arbitrary PPT adversary against the position binding property of the construction. By applying the definition of the weak verification algorithm and finally splitting the probability depending on $\bar{\boldsymbol{m}}_0 = \bar{\boldsymbol{m}}_1$ we obtain

$$
\begin{aligned}
&\Pr\left[\begin{array}{c}\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda, \tau);\\ (\boldsymbol{c}, t, (\bar{\boldsymbol{d}}_0, \boldsymbol{b}_0), (\bar{\boldsymbol{d}}_1, \boldsymbol{b}_1)) \leftarrow \mathcal{A}(\mathsf{pp});\\ \boldsymbol{m}_0 \leftarrow \mathsf{wVrfy}(\mathsf{pp}, c, t, (\bar{\boldsymbol{d}}_0, \boldsymbol{b}_0));\\ \boldsymbol{m}_1 \leftarrow \mathsf{wVrfy}(\mathsf{pp}, c, t, (\bar{\boldsymbol{d}}_1, \boldsymbol{b}_1))\end{array} : \begin{array}{c}\boldsymbol{m}_0 \neq \boldsymbol{m}_1\\ \wedge \perp \notin \{\boldsymbol{m}_0, \boldsymbol{m}_1\}\end{array}\right]\\
=&\Pr\left[\begin{array}{c}\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda, \tau);\\ (\boldsymbol{c}, t, (\bar{\boldsymbol{d}}_0, \boldsymbol{b}_0), (\bar{\boldsymbol{d}}_1, \boldsymbol{b}_1)) \leftarrow \mathcal{A}(\mathsf{pp});\\ \bar{\boldsymbol{m}}_0 \leftarrow \overline{\mathsf{wVrfy}}(\overline{\mathsf{pp}}, c, t, \bar{\boldsymbol{d}}_0);\\ \bar{\boldsymbol{m}}_1 \leftarrow \overline{\mathsf{wVrfy}}(\overline{\mathsf{pp}}, c, t, \bar{\boldsymbol{d}}_1)\end{array} : \begin{array}{c}\|\boldsymbol{b}_0\| \leq 2\rho\alpha \wedge \|\boldsymbol{b}_1\| \leq 2\rho\alpha\\ \wedge \bar{\boldsymbol{m}}_0 = \boldsymbol{h}^\intercal \cdot \boldsymbol{b}_0 \wedge \bar{\boldsymbol{m}}_1 = \boldsymbol{h}^\intercal \cdot \boldsymbol{b}_1\\ \wedge \mathsf{proj}(\boldsymbol{b}_0) \neq \mathsf{proj}(\boldsymbol{b}_1)\end{array}\right]\\
=&\Pr\left[\begin{array}{c}\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda, \tau);\\ (\boldsymbol{c}, t, (\bar{\boldsymbol{d}}_0, \boldsymbol{b}_0), (\bar{\boldsymbol{d}}_1, \boldsymbol{b}_1)) \leftarrow \mathcal{A}(\mathsf{pp});\\ \bar{\boldsymbol{m}}_0 \leftarrow \overline{\mathsf{wVrfy}}(\overline{\mathsf{pp}}, c, t, \bar{\boldsymbol{d}}_0);\\ \bar{\boldsymbol{m}}_1 \leftarrow \overline{\mathsf{wVrfy}}(\overline{\mathsf{pp}}, c, t, \bar{\boldsymbol{d}}_1)\end{array} : \begin{array}{c}\|\boldsymbol{b}_0\| \leq 2\rho\alpha \wedge \|\boldsymbol{b}_1\| \leq 2\rho\alpha\\ \wedge \bar{\boldsymbol{m}}_0 = \boldsymbol{h}^\intercal \cdot \boldsymbol{b}_0 \wedge \bar{\boldsymbol{m}}_1 = \boldsymbol{h}^\intercal \cdot \boldsymbol{b}_1\\ \wedge \mathsf{proj}(\boldsymbol{b}_0) \neq \mathsf{proj}(\boldsymbol{b}_1) \wedge \bar{\boldsymbol{m}}_0 \neq \bar{\boldsymbol{m}}_1\end{array}\right] \qquad (5)\\
&+\Pr\left[\begin{array}{c}\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda, \tau);\\ (\boldsymbol{c}, t, (\bar{\boldsymbol{d}}_0, \boldsymbol{b}_0), (\bar{\boldsymbol{d}}_1, \boldsymbol{b}_1)) \leftarrow \mathcal{A}(\mathsf{pp});\\ \bar{\boldsymbol{m}}_0 \leftarrow \overline{\mathsf{wVrfy}}(\overline{\mathsf{pp}}, c, t, \bar{\boldsymbol{d}}_0);\\ \bar{\boldsymbol{m}}_1 \leftarrow \overline{\mathsf{wVrfy}}(\overline{\mathsf{pp}}, c, t, \bar{\boldsymbol{d}}_1)\end{array} : \begin{array}{c}\|\boldsymbol{b}_0\| \leq 2\rho\alpha \wedge \|\boldsymbol{b}_1\| \leq 2\rho\alpha\\ \wedge \bar{\boldsymbol{m}}_0 = \boldsymbol{h}^\intercal \cdot \boldsymbol{b}_0 \wedge \bar{\boldsymbol{m}}_1 = \boldsymbol{h}^\intercal \cdot \boldsymbol{b}_1\\ \wedge \mathsf{proj}(\boldsymbol{b}_0) \neq \mathsf{proj}(\boldsymbol{b}_1) \wedge \bar{\boldsymbol{m}}_0 = \bar{\boldsymbol{m}}_1\end{array}\right]. \qquad (6)
\end{aligned}
$$

It remains to bound the two probabilities seperately. To bound Equation 5 we construct an adversary $\mathcal{B}$ against the position binding of $\overline{\mathsf{HVC}}$ as follows. Upon input $\overline{\mathsf{pp}}$, $\mathcal{B}$ samples $\boldsymbol{h} \leftarrow \mathcal{R}_{\bar{q}}^{\xi\lceil \log q\rceil}$, invokes $(\boldsymbol{c}, t, (\bar{\boldsymbol{d}}_0, \boldsymbol{b}_0), (\bar{\boldsymbol{d}}_1, \boldsymbol{b}_1)) \leftarrow \mathcal{A}((\overline{\mathsf{pp}}, \boldsymbol{h}))$ and outputs $(\boldsymbol{c}, t, \bar{\boldsymbol{d}}_0, \bar{\boldsymbol{d}}_1)$. Since $\overline{\mathsf{HVC}}$ is position binding, it holds that

$$
\begin{aligned}
\mathsf{negl}(\lambda) \geq &\Pr\left[\begin{array}{c}\overline{\mathsf{pp}} \leftarrow \overline{\mathsf{Setup}}(1^\lambda, \tau);\\ (\boldsymbol{c}, t, \bar{\boldsymbol{d}}_0, \bar{\boldsymbol{d}}_1) \leftarrow \mathcal{B}(\mathsf{pp});\\ \bar{\boldsymbol{m}}_0 \leftarrow \overline{\mathsf{wVrfy}}(\overline{\mathsf{pp}}, c, t, \bar{\boldsymbol{d}}_0);\\ \bar{\boldsymbol{m}}_1 \leftarrow \overline{\mathsf{wVrfy}}(\overline{\mathsf{pp}}, c, t, \bar{\boldsymbol{d}}_1)\end{array} : \begin{array}{c}\bar{\boldsymbol{m}}_0 \neq \bar{\boldsymbol{m}}_1\\ \wedge \perp \notin \{\bar{\boldsymbol{m}}_0, \bar{\boldsymbol{m}}_1\}\end{array}\right]\\
\geq &\Pr\left[\begin{array}{c}\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda, \tau);\\ (\boldsymbol{c}, t, (\bar{\boldsymbol{d}}_0, \boldsymbol{b}_0), (\bar{\boldsymbol{d}}_1, \boldsymbol{b}_1)) \leftarrow \mathcal{A}(\mathsf{pp});\\ \bar{\boldsymbol{m}}_0 \leftarrow \overline{\mathsf{wVrfy}}(\overline{\mathsf{pp}}, c, t, \bar{\boldsymbol{d}}_0);\\ \bar{\boldsymbol{m}}_1 \leftarrow \overline{\mathsf{wVrfy}}(\overline{\mathsf{pp}}, c, t, \bar{\boldsymbol{d}}_1)\end{array} : \begin{array}{c}\bar{\boldsymbol{m}}_0 \neq \bar{\boldsymbol{m}}_1\\ \wedge \perp \notin \{\bar{\boldsymbol{m}}_0, \bar{\boldsymbol{m}}_1\}\end{array}\right]\\
\geq &\Pr\left[\begin{array}{c}\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda, \tau);\\ (\boldsymbol{c}, t, (\bar{\boldsymbol{d}}_0, \boldsymbol{b}_0), (\bar{\boldsymbol{d}}_1, \boldsymbol{b}_1)) \leftarrow \mathcal{A}(\mathsf{pp});\\ \bar{\boldsymbol{m}}_0 \leftarrow \overline{\mathsf{wVrfy}}(\overline{\mathsf{pp}}, c, t, \bar{\boldsymbol{d}}_0);\\ \bar{\boldsymbol{m}}_1 \leftarrow \overline{\mathsf{wVrfy}}(\overline{\mathsf{pp}}, c, t, \bar{\boldsymbol{d}}_1)\end{array} : \begin{array}{c}\|\boldsymbol{b}_0\| \leq 2\rho\alpha \wedge \|\boldsymbol{b}_1\| \leq 2\rho\alpha\\ \wedge \bar{\boldsymbol{m}}_0 = \boldsymbol{h}^\intercal \cdot \boldsymbol{b}_0 \wedge \bar{\boldsymbol{m}}_1 = \boldsymbol{h}^\intercal \cdot \boldsymbol{b}_1\\ \wedge \mathsf{proj}(\boldsymbol{b}_0) \neq \mathsf{proj}(\boldsymbol{b}_1) \wedge \bar{\boldsymbol{m}}_0 \neq \bar{\boldsymbol{m}}_1\end{array}\right]
\end{aligned}
$$

To bound Equation 6, we construct a PPT algorithm that solves the $\mathsf{SIS}_{\mathcal{R},\bar{q},\xi\lceil \log q\rceil,4\rho\alpha}$ problem as follows. Upon input $\boldsymbol{h} \in \mathcal{R}_{\bar{q}}^{\xi\lceil \log q\rceil}$, $\mathcal{C}$ runs $\bar{\mathsf{p}} \leftarrow \overline{\mathsf{Setup}}(1^\lambda, \tau)$ and invokes $(\boldsymbol{c}, t, (\bar{\boldsymbol{d}}_0, \boldsymbol{b}_0), (\bar{\boldsymbol{d}}_1, \boldsymbol{b}_1)) \leftarrow \mathcal{A}((\overline{\mathsf{pp}}), \boldsymbol{h})$. Finally it outputs $\boldsymbol{b}_0 - \boldsymbol{b}_1$. Since $\mathsf{SIS}_{\mathcal{R},\bar{q},\xi\lceil \log q\rceil,4\rho\alpha}$ is hard, it holds that

$$
\mathsf{negl}(\lambda) \geq \Pr[\boldsymbol{h} \leftarrow \mathcal{R}_{\bar{q}}^{\xi\lceil \log q\rceil}; \boldsymbol{s} \leftarrow \mathcal{C}(\boldsymbol{h}) : \boldsymbol{s} \in \mathcal{B}_{4\rho\alpha}^{\xi\lceil \log q\rceil} \setminus \{\boldsymbol{0}\} \wedge \boldsymbol{h}^\intercal \boldsymbol{s} = 0]
$$

$$
\begin{aligned}
&\geq \Pr\left[\begin{array}{c} \boldsymbol{h} \leftarrow \mathcal{R}_{\bar{q}}^{\xi\lceil \log q\rceil}; \\ (\boldsymbol{b}_0 - \boldsymbol{b}_1) \leftarrow \mathcal{C}(\boldsymbol{h}) \end{array} : \begin{array}{c} \|\boldsymbol{b}_0\| \leq 2\rho\alpha \wedge \|\boldsymbol{b}_1\| \leq 2\rho\alpha \\ \wedge \boldsymbol{h}^\intercal \boldsymbol{b}_0 = \boldsymbol{h}^\intercal \boldsymbol{b}_1 \wedge \boldsymbol{b}_0 \neq \boldsymbol{b}_1 \end{array}\right] \\
&= \Pr\left[\begin{array}{c} \mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda, \tau); \\ (\boldsymbol{c}, t, (\bar{\boldsymbol{d}}_0, \boldsymbol{b}_0), (\bar{\boldsymbol{d}}_1, \boldsymbol{b}_1)) \leftarrow \mathcal{A}(\mathsf{pp}); \end{array} : \begin{array}{c} \|\boldsymbol{b}_0\| \leq 2\rho\alpha \wedge \|\boldsymbol{b}_1\| \leq 2\rho\alpha \\ \wedge \boldsymbol{h}^\intercal \cdot \boldsymbol{b}_0 = \boldsymbol{h}^\intercal \cdot \boldsymbol{b}_1 \wedge \boldsymbol{b}_0 \neq \boldsymbol{b}_1 \end{array}\right] \\
&\geq \Pr\left[\begin{array}{c} \mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda, \tau); \\ (\boldsymbol{c}, t, (\bar{\boldsymbol{d}}_0, \boldsymbol{b}_0), (\bar{\boldsymbol{d}}_1, \boldsymbol{b}_1)) \leftarrow \mathcal{A}(\mathsf{pp}); \\ \bar{\boldsymbol{m}}_0 \leftarrow \overline{\mathsf{wVrfy}}(\overline{\mathsf{pp}}, c, t, \bar{\boldsymbol{d}}_0); \\ \bar{\boldsymbol{m}}_1 \leftarrow \overline{\mathsf{wVrfy}}(\overline{\mathsf{pp}}, c, t, \bar{\boldsymbol{d}}_1) \end{array} : \begin{array}{c} \|\boldsymbol{b}_0\| \leq 2\rho\alpha \wedge \|\boldsymbol{b}_1\| \leq 2\rho\alpha \\ \wedge \bar{\boldsymbol{m}}_0 = \boldsymbol{h}^\intercal \cdot \boldsymbol{b}_0 \wedge \bar{\boldsymbol{m}}_1 = \boldsymbol{h}^\intercal \cdot \boldsymbol{b}_1 \\ \wedge \mathsf{proj}(\boldsymbol{b}_0) \neq \mathsf{proj}(\boldsymbol{b}_1) \wedge \bar{\boldsymbol{m}}_0 = \bar{\boldsymbol{m}}_1 \end{array}\right]
\end{aligned}
$$

Since both probabilities are thus bounded by negligible functions, the lemma follows. □

4 Key-Homomorphic One-Time Signatures

In this section, we define and instantiate the notion of a key-homomorphic one-time signature scheme that we will need in our final construction. Intuitively, a one-time signature is unforgeable as long as at most one signature for some message is published under a given public key. We call such a scheme homomorphic, if the a linear combination of separate signatures for the same message verifies under the linear combination of the corresponding public keys, while still being unforgeable. We present a construction of this primitive, which is similar to previous one-time signature schemes by Boneh and Kim [BK20] and Lyubashevsky and Micciancio [LM08].

Definition 8 (One-Time Signature). *Let $\mathcal{R}$ be a ring. A key-homomorphic one-time signature scheme (KOTS) over $\mathcal{R}$ with public key length ℓ_{opk} and signature length ℓ_{sig} is defined by four PPT algorithms* $\mathsf{KOTS} = (\mathsf{Setup}, \mathsf{KGen}, \mathsf{Sign}, \mathsf{Vrfy})$.

$\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda)$ *The setup algorithm takes as input the security parameter and outputs public parameters.*

$(\mathsf{osk}, \mathsf{opk}) \leftarrow \mathsf{KGen}(\mathsf{pp})$ *The key generation algorithm takes as input the public parameters and outputs a key pair with* $\mathsf{opk} \in \mathcal{R}_q^{\ell_{\mathsf{opk}}}$.

$\sigma \leftarrow \mathsf{Sign}(\mathsf{pp}, \mathsf{osk}, m)$ *The signing algorithm takes as input the public parameters, a one-time signing key, and a message and outputs a signature* $\sigma \in \mathcal{R}_q^{\ell_{\mathsf{sig}}}$.

$b \leftarrow \mathsf{wVrfy}(\mathsf{pp}, \mathsf{opk}, m, \sigma)$ *The weak verification algorithm takes as input the public parameters, a verification key, a message, and a candidate signature and outputs a bit indicating acceptance/rejection.*

$b \leftarrow \mathsf{sVrfy}(\mathsf{pp}, \mathsf{opk}, m, \sigma)$ *The strong verification algorithm takes as input the public parameters, a verification key, a message, and a candidate signature and outputs a bit indicating acceptance/rejection.*

A one-time signature is (ρ, W)-homomorphically correct, if for all security parameters $\lambda \in \mathbb{N}$, $\ell \in [\rho]$, messages $m \in \{0,1\}^*$, and ring elements $w_1, \ldots, w_\ell \in W$ it holds that

$$
\Pr\left[\begin{array}{c} \mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda); \\ (\mathsf{osk}^i, \mathsf{opk}^i) \leftarrow \mathsf{KGen}(\mathsf{pp}); \\ \sigma^i \leftarrow \mathsf{Sign}(\mathsf{pp}, \mathsf{osk}^i, m) \end{array} : \mathsf{sVrfy}(\mathsf{pp}, \sum_{i=1}^{\ell} w^i \cdot \mathsf{opk}^i, m, \sum_{i=1}^{\ell} w^i \cdot \sigma^i) = 1\right] = 1
$$

Remark 2. Note that again the homomorphic correctness definition above implies regular correctness of unaggregated signatures with $\ell = 1$ and $W = \{1\}$.

As with the vector commitments from the previous section, we want our signature scheme to be robustly homomorphic in the sense that the difference of two maliciously generated signatures under malicious public keys will verify, if the individual signatures verify.

Definition 9. *Let* KOTS *be a* (ρ, W)*-homomorphically correct one-time signature scheme over* $\mathcal{R}$ *with public key length* ℓ_{opk} *and signature length* ℓ_{sig}*.* KOTS *is robustly homomorphic if for all* $\lambda \in \mathbb{N}$*,* $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda)$*,* $m \in \{0,1\}^*$*,* $\mathsf{opk}^0, \mathsf{opk}^1 \in \mathcal{R}_q^{\ell_{\mathsf{opk}}}$*, and* $\sigma^0, \sigma^1 \in \mathcal{R}_q^{\ell_{\mathsf{sig}}}$ *such that*

$$\mathsf{sVrfy}(\mathsf{pp}, \mathsf{opk}^0, m, \sigma^0) = 1 \quad \textit{and} \quad \mathsf{sVrfy}(\mathsf{pp}, \mathsf{opk}^1, m, \sigma^1) = 1$$

it holds that

$$\mathsf{wVrfy}(\mathsf{pp}, \mathsf{opk}^0 - \mathsf{opk}^1, m, (\sigma^0 - \sigma^1)) = 1.$$

We define a multi-user version of (one-time) existential unforgeability, this will allow for a tighter proof of the synchronized multi-signature scheme. The definition is further strengthened by allowing the adversary to produce forgeries not just under one of the given public keys, but also under mildly rerandomized public key.

Definition 10 (Multi-User Existential Unforgeability under Rerandomized Keys). *A* (ρ, W)*-homomorphically correct KOTS is* W'*-existentially unforgeable under rerandomized keys (EUF-RK), if for all security parameters* λ*, any* $T = \mathsf{poly}(\lambda)(\lambda) \in \mathbb{N}$ *and all stateful PPT algorithms* $\mathcal{A}$ *it holds that*

$$\Pr\left[\begin{array}{r} \mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda); \\ \forall i \in [T-1].\, (\mathsf{osk}_i, \mathsf{opk}_i) \leftarrow \mathsf{KGen}(\mathsf{pp}); \\ (i^*, m^*, \sigma^*, w^*) \leftarrow \mathcal{A}^{\widetilde{\mathsf{Sign}}(\cdot,\cdot)}(\mathsf{pp}, \mathsf{opk}_0, \ldots, \mathsf{opk}_{T-1}); \end{array} : \begin{array}{l} \mathsf{wVrfy}(\mathsf{pp}, w^* \cdot \mathsf{opk}_{i^*}, m^*, \sigma^*) = 1 \\ \wedge m^* \notin Q_i \wedge |Q_i| \leq 1 \wedge w^* \in W' \end{array}\right] \leq \mathsf{negl}(\lambda),$$

where the oracle $\widetilde{\mathsf{Sign}}(\cdot,\cdot)$ *is defined as* $\widetilde{\mathsf{Sign}}(i, m) := \mathsf{Sign}(\mathsf{osk}_i, m)$ *and* Q_i *denotes the set of messages for which a signing query with index* i *has been made.*

Our construction presented here closely follows a construction that appeared previously in the work of Boneh and Kim [BK20].

Theorem 14. *Let* $\mathcal{R}_q = \mathbb{Z}_q[x]/\langle x^n + 1\rangle$ *be a polynomial ring parameterized by* $n = \mathsf{poly}(\lambda)$ *and* $q = \mathsf{poly}(\lambda)$*. Let* α *be the smallest integer, such that* $\binom{n}{\alpha} \cdot 2^\alpha \geq 2^\lambda$*. Let* $W' = \{w_0 - w_1 \mid w_0, w_1 \in \mathcal{T}_\alpha \wedge w_0 \neq w_1\}$*. If the* $\mathsf{SIS}_{\mathcal{R},q,\gamma,(4\rho+4)\alpha\beta_s}$ *problem is hard and* $H : \{0,1\}^* \to \mathcal{T}_{\beta_s}$ *is collision resistant, then the construction from Figure 3 is a* (ρ, W)*-homomorphically correct KOTS that is multi-user existentially unforgeable under rerandomized keys.*

Proof. The theorem follows from Lemma 15, Lemma 16, and Lemma 17.

The following three lemmas state that our construction satisfies the desired homomorphic properties and that it is unforgeable.

Lemma 15. *Let* $\beta_s, \alpha, \rho \in \mathbb{N}$ *and let* $H : \{0,1\}^* \to \mathcal{T}_{\beta_s}\}$ *be a hash function. Let* $\beta_\sigma = 2\rho\alpha\beta_s$*. The construction from Figure 3 is a* $(\rho, \mathcal{T}_\alpha)$*-homomorphically correct one time signature scheme.*

$\mathsf{Setup}(1^\lambda)$	$\mathsf{KGen}(\mathsf{pp})$	$\mathsf{Sign}(\mathsf{pp}, \mathsf{osk}, m)$
$\boldsymbol{a} \leftarrow \mathcal{R}_q^\gamma$	$\boldsymbol{s}_0 \leftarrow \mathcal{B}_1^\gamma$	**parse** osk **as** $(\boldsymbol{s}_0, \boldsymbol{s}_1)$
return $\boldsymbol{a}$	$\boldsymbol{s}_1 \leftarrow \mathcal{B}_{\beta_s}^\gamma$	$\sigma := \boldsymbol{s}_0 \cdot H(m) + \boldsymbol{s}_1$
	$v_0 := \boldsymbol{a}^\intercal \cdot s_0$	**return** σ
	$v_1 := \boldsymbol{a}^\intercal \cdot s_1$	
	return $((\boldsymbol{s}_0, \boldsymbol{s}_1)(v_0, v_1))$	

$\mathsf{wVrfy}(\mathsf{pp}, \mathsf{opk}, m, \sigma)$	$\mathsf{Vrfy}(\mathsf{pp}, \mathsf{opk}, m, \sigma, \beta')$
return $\mathsf{Vrfy}(\mathsf{pp}, \mathsf{opk}, m, \sigma, 2\beta_\sigma)$	**parse** opk **as** (v_0, v_1)
	if $\|\sigma\| > \beta'$
$\mathsf{sVrfy}(\mathsf{pp}, \mathsf{opk}, m, \sigma)$	**return** 0
return $\mathsf{Vrfy}(\mathsf{pp}, \mathsf{opk}, m, \sigma, \beta_\sigma)$	**if** $\boldsymbol{a}^\intercal \cdot \sigma \neq v_0 \cdot H(m) + v_1$
	return 0
	return 1

Fig. 3. Description of the key-homomorphic one-time signature scheme. H is a collision-resistant hash function mapping bit-strings to $\mathcal{T}_{\beta_s}$.

Proof. Let $\lambda \in \mathbb{N}$, $\ell \in [\rho]$, $m \in \{0,1\}^*$, and $w_1, \ldots, w_\ell \in \mathcal{T}_\alpha$ as in Definition 8. Let $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda)$ and for $i \in [\ell]$, let $(\mathsf{osk}^i, \mathsf{opk}^i) \leftarrow \mathsf{KGen}(\mathsf{pp})$ and $\sigma^i = \mathsf{Sign}(\mathsf{pp}, \mathsf{osk}^i, m)$. We need to show that both checks in the verification algorithm go through for $\beta' = \beta_\sigma$. We first observe that the norm check goes through:

$$\begin{aligned}
\left\|\sum_{i=0}^{\ell-1} w^i \cdot \sigma^i\right\| &= \left\|\sum_{i=0}^{\ell-1} w^i \cdot (\boldsymbol{s}_0^i \cdot H(m) + \boldsymbol{s}_1^i)\right\| && \text{(Def of } \mathsf{Sign}\text{)} \\
&\leq \sum_{i=0}^{\ell-1} \left\|w^i \cdot (\boldsymbol{s}_0^i \cdot H(m) + \boldsymbol{s}_1^i)\right\| && \text{(Triangle Inequality)} \\
&\leq \sum_{i=0}^{\ell-1} \alpha \cdot (\left\|\boldsymbol{s}_0^i \cdot H(m)\right\| + \left\|\boldsymbol{s}_1^i\right\|) && \text{(Lemma 1)} \\
&\leq \sum_{i=0}^{\ell-1} \alpha \cdot (\beta_s + \beta_s) && \text{(Lemma 1)} \\
&= 2\ell\alpha\beta_s \leq 2\rho\alpha\beta_s = \beta_\sigma.
\end{aligned}$$

It remains to verify that the second check goes through:

$$\begin{aligned}
\boldsymbol{a}^\intercal \cdot \sum_{i=0}^{\ell-1} w^i \cdot \sigma^i &= \boldsymbol{a}^\intercal \cdot \sum_{i=0}^{\ell-1} w^i \cdot (\boldsymbol{s}_0^i \cdot H(m) + \boldsymbol{s}_1^i) && \text{(Def of } \mathsf{Sign}\text{)} \\
&= \sum_{i=0}^{\ell-1} w^i \cdot (\boldsymbol{a}^\intercal \cdot \boldsymbol{s}_0^i \cdot H(m) + \boldsymbol{a}^\intercal \cdot \boldsymbol{s}_1^i) && \text{(Distributivity)} \\
&= \sum_{i=0}^{\ell-1} w^i \cdot (v_0^i \cdot H(m) + v_1^i) && \text{(Def of } \mathsf{KGen}\text{)}
\end{aligned}$$

$$= \Big(\sum_{i=0}^{\ell-1} w^i \cdot v_0^i\Big) \cdot H(m) + \Big(\sum_{i=1}^{\ell} w^i \cdot v_1^i\Big)$$

The lemma statement thus follows. □

Lemma 16. *Let $\beta_s \in \mathbb{N}$ and let $H : \{0,1\}^* \to \mathcal{T}_{\beta_s}$ be a hash function. Then the construction from Figure 3 is a robustly homomorphic.*

Proof. Let $\lambda \in \mathbb{N}$, $\mathsf{p} \leftarrow \mathsf{Setup}(1^\lambda)$, $m \in \{0,1\}^*$, $\mathsf{opk}^0, \mathsf{opk}^1 \in \mathcal{R}_q$, and $\sigma^0, \sigma^1 \in \mathcal{R}_q^\gamma$ as specified in Definition 9. To conclude that the scheme is robustly homomorphic we need to verify that both checks in the verification algorithm go through for $\beta' = 2\beta_\sigma$.

Since both σ^0 and σ^1 *strongly* verify, it holds that

$$\big\|(\sigma^0 - \sigma^1)\big\| \leq \big\|\sigma^0\big\| + \big\|\sigma^1\big\| \leq 2\beta_\sigma,$$

thus the norm check goes through. It remains to verify that the second check also goes through.

$$\begin{aligned} \boldsymbol{a}^\intercal \cdot (\sigma^0 - \sigma^1) =& \boldsymbol{a}^\intercal \cdot \sigma^0 - \boldsymbol{a}^\intercal \cdot \sigma^1 \\ =& (v_0^0 \cdot H(m) + v_1^0) - (v_0^1 \cdot H(m) + v_1^1) && \text{(Def of } \mathsf{sVrfy}) \\ =& (v_0^0 - v_0^1) \cdot H(m) + (v_1^0 - v_1^1). \end{aligned}$$

Therefore, the lemma statement follows. □

Lemma 17. *Let $n, \gamma, q, \beta_s, \alpha, \delta$ be positive integers with q prime and n a power of 2, such that $q > 16\alpha\beta_s$, $2^{(3\lambda+\delta)/n\gamma} \cdot q^{1/\gamma} \leq 3/2$, and $2^{2\lambda} \leq |\mathcal{T}_{\beta_s}| \leq 2^{2\lambda+\delta}$. Let $H : \{0,1\}^* \to \mathcal{T}_{\beta_s}$ be a hash function. Let $\beta_\sigma = 2\rho\alpha\beta_s$. If the $\mathsf{SIS}_{\mathcal{R},q,\gamma,(4\rho+4)\alpha\beta_s}$ problem is hard and H is collision resistant, then the construction from Figure 3 is existentially unforgeable under rerandomized keys.*

Proof. Let $\mathcal{A}$ be an arbitrary adversary against the multi-user existentially unforgeability under rerandomized keys with success probability $\epsilon = \epsilon(\lambda)$. We construct an algorithm $\mathcal{B}$ that solves $(\mathcal{R}, q, \gamma, (4\rho+8)\alpha\beta_s)$-SIS as follows. Given $\boldsymbol{a} \in \mathcal{R}_q^\gamma$, $\mathcal{B}$ chooses secret keys $(\boldsymbol{s}_0^i, \boldsymbol{s}_1^i) \in \mathcal{B}_1^\gamma \times \mathcal{B}_{\beta_s}^\gamma$ uniformly at random for $i \in [T-1]$ and invokes $\mathcal{A}$ on public keys v_0^i, v_1^i, with $v_b^i := \boldsymbol{a}^\intercal \cdot s_b^i$. Whenever $\mathcal{A}$ sends a signing query i, m, $\mathcal{B}$ will respond by sending the honestly computed signature $\sigma := \boldsymbol{s}_0^i \cdot H(m) + \boldsymbol{s}_1^i$. Eventually $\mathcal{A}$ will then output a supposed forgery $(i^*, m^*, \sigma^*, w^*)$ and $\mathcal{B}$ will compute the signature on the same message as $\sigma' := w^* \cdot \boldsymbol{s}_0^{i^*} \cdot H(m^*) + w^* \cdot \boldsymbol{s}_1^{i^*}$. It then outputs $\sigma^* - \sigma'$.

To analyze the success probability of $\mathcal{B}$, suppose that $\mathcal{A}$ outputs a valid forgery. We first note that, *if* $\sigma^* \neq \sigma'$, it holds that $\sigma^* - \sigma' \neq 0$ and further, since both signatures verify, that

$$\begin{aligned} \boldsymbol{a}^\intercal \cdot (\sigma^* - \sigma') =& \boldsymbol{a}^\intercal \cdot \sigma^* - \boldsymbol{a}^\intercal \cdot \sigma' \\ =& (w^* \cdot v_0^{i^*} \cdot H(m) + w^* \cdot v_1^{i^*}) - (w^* \cdot v_0^{i^*} \cdot H(m) + w^* \cdot v_1^{i^*}) = 0. \end{aligned}$$

Since σ^* weakly verifies, it must hold that $\|\sigma^*\| \leq 2\beta_\sigma$. Further, σ' is an honestly computed signature for a secret key rerandomized with $w^* \in W'$. Therefore, there exist $w_0, w_1 \in \mathcal{T}_\alpha$, such that $w^* = w_0 - w_1$ and it holds that

$$\big\|\sigma'\big\| = \Big\|w^* \cdot \boldsymbol{s}_0^{i^*} \cdot H(m^*) + w^* \cdot \boldsymbol{s}_1^{i^*}\Big\| \qquad \text{(Def. of } \mathsf{Sign})$$

$$
\begin{aligned}
&= \left\| (w_0 - w_1) \cdot \boldsymbol{s}_0^{i^*} \cdot H(m^*) + (w_0 - w_1) \cdot \boldsymbol{s}_1^{i^*} \right\| && (w^* \in W') \\
&= \left\| w_0 \cdot (\boldsymbol{s}_0^{i^*} \cdot H(m^*) + \boldsymbol{s}_1^{i^*}) - w_1 \cdot (\boldsymbol{s}_0^{i^*} \cdot H(m^*) + \boldsymbol{s}_1^{i^*}) \right\| && (\text{Distributivity}) \\
&\leq \left\| w_0 \cdot (\boldsymbol{s}_0^{i^*} \cdot H(m^*) + \boldsymbol{s}_1^{i^*}) \right\| + \left\| w_1 \cdot (\boldsymbol{s}_0^{i^*} \cdot H(m^*) + \boldsymbol{s}_1^{i^*}) \right\| && (\text{Triangle Inequality}) \\
&\leq 2\alpha \cdot \left(\left\| \boldsymbol{s}_0^{i^*} \cdot H(m^*) \right\| + \left\| \boldsymbol{s}_1^{i^*} \right\| \right) && (\text{Lemma 1}) \\
&= 4\alpha\beta_s && (\text{Lemma 1})
\end{aligned}
$$

It thus holds that $\|\sigma^* - \sigma'\| \leq \|\sigma^*\| + \|\sigma'\| \leq 2 \cdot \beta_\sigma + 4\alpha\beta_s = 4\rho\alpha\beta_s + 4\alpha\beta_s = (4\rho + 4)\alpha\beta_s$. Thus $\sigma^* - \sigma'$ is always a valid solution to the SIS-instance. Thus, it remains to bound the probability, that $\sigma^* = \sigma'$.

For this, we observe by Lemma 18 that the secret key for which $\mathcal{A}$ chooses to forge is information-theoretically hidden from $\mathcal{A}$ among at least 2 possible secret keys. Once $\mathcal{A}$ outputs a valid forgery $(i^*, m^*, \sigma^*, w^*)$, the signing key used for the forgery becomes uniquely determined by Lemma 19 as long as $H(m^*) \neq H(m)$, which is guaranteed with overwhelming probability by the collision resistance of H. It follows that $\sigma^* \neq \sigma'$ with probability at least $1/2 - \mathsf{negl}(\lambda)$. Therefore, the success probability of our reduction $\mathcal{B}$ is $(1/2 - \mathsf{negl}(\lambda))\epsilon$ and since the SIS problem is hard, ϵ must be negligible in λ. □

Lemma 18. *Let $n, \gamma, q, \beta_s, \delta$ be positive integers such that $2^{(3\lambda+\delta)/n\gamma} \cdot q^{1/\gamma} \leq 3/2$ and $2^{2\lambda} \leq |\mathcal{T}_{\beta_s}| \leq 2^{2\lambda+\delta}$, let $\mathcal{R} = \mathbb{Z}[x]/(x^n + 1)$. Then for any $\boldsymbol{a} \in \mathcal{R}_q^\gamma$ and uniformly chosen $(\boldsymbol{s}_0, \boldsymbol{s}_1) \in \mathcal{B}_1^\gamma \times \mathcal{B}_{\beta_s}^\gamma$ it holds with probability at least $1 - 2^{-\lambda}$ that for every $c \in \mathcal{T}_{\beta_s}$ there exists $(\boldsymbol{s}_0', \boldsymbol{s}_1') \in \mathcal{B}_1^\gamma \times \mathcal{B}_{\beta_s}^\gamma$ such that $(\boldsymbol{s}_0', \boldsymbol{s}_1') \neq (\boldsymbol{s}_0, \boldsymbol{s}_1)$, $(\boldsymbol{a}^\intercal \cdot \boldsymbol{s}_0', \boldsymbol{a}^\intercal \cdot \boldsymbol{s}_1') = (\boldsymbol{a}^\intercal \cdot \boldsymbol{s}_0, \boldsymbol{a}^\intercal \cdot \boldsymbol{s}_1)$ and $\boldsymbol{s}_0' \cdot c + \boldsymbol{s}_1' = \boldsymbol{s}_0 \cdot c + \boldsymbol{s}_1$.*

Proof. Our proof closely follows the proof from [LM08, Lemma 4.9]. We define a function $f_{\boldsymbol{a},c}$ that maps any secret key $(\boldsymbol{s}_0, \boldsymbol{s}_1)$ to a pair of the public key and signature of message c defined as $((\boldsymbol{a}^\intercal \cdot \boldsymbol{s}_0, \boldsymbol{a}^\intercal \cdot \boldsymbol{s}_1), \boldsymbol{s}_0 \cdot c + \boldsymbol{s}_1)$. We will show that the domain of this function is at least $2^{3\lambda+\delta}$ times larger than the range. The number of possible secret keys is $3^{n\gamma} \cdot (2\beta_s + 1)^{n\gamma}$. The number of possible signatures is at most $(4\beta_s + 1)^{n\gamma}$. For fixed values $\boldsymbol{a}, c, \boldsymbol{s}_0 \cdot c + \boldsymbol{s}_1$, we observe that once $\boldsymbol{a}^\intercal \cdot \boldsymbol{s}_0$ is fixed, the second component $\boldsymbol{a}^\intercal \cdot \boldsymbol{s}_1 = \boldsymbol{a}^\intercal \cdot ((\boldsymbol{s}_0 \cdot c + \boldsymbol{s}_1) - s_0 \cdot c)$ is uniquely determined. Thus for a fixed signature, there are at most q^n many possible public keys and therefore the range of $f_{\boldsymbol{a},c}$ is at most $(4\beta_s + 1)^{n\gamma} \cdot q^n$. We observe that

$$
\frac{3^{n\gamma} \cdot (2\beta_s + 1)^{n\gamma}}{(4\beta_s + 1)^{n\gamma} \cdot q^n} \geq \frac{3^{n\gamma} \cdot (2\beta_s + 1)^{n\gamma}}{(4\beta_s + 2)^{n\gamma} \cdot q^n} = \frac{3^{n\gamma}}{2^{n\gamma} \cdot q^n}
$$

Using the inequality from the lemma statement, one can see that

$$
2^{(3\lambda+\delta)/n\gamma} \cdot q^{1/\gamma} \leq \frac{3}{2} \implies 2^{3\lambda+\delta} \cdot q^n \leq \left(\frac{3}{2}\right)^{n\gamma} \implies 2^{3\lambda+\delta} \leq \frac{3^{n\gamma}}{2^{n\gamma} \cdot q^n}
$$

Using Lemma 4.1 from [LM08], the probability, over a uniformly chosen secret key, that there exists $(\boldsymbol{s}_0', \boldsymbol{s}_1') \in \mathcal{B}_1^\gamma \times \mathcal{B}_{\beta_s}^\gamma$ such that $(\boldsymbol{s}_0', \boldsymbol{s}_1') \neq (\boldsymbol{s}_0, \boldsymbol{s}_1)$, $(\boldsymbol{a}^\intercal \cdot \boldsymbol{s}_0', \boldsymbol{a}^\intercal \cdot \boldsymbol{s}_1') = (\boldsymbol{a}^\intercal \cdot \boldsymbol{s}_0, \boldsymbol{a}^\intercal \cdot \boldsymbol{s}_1)$ and $\boldsymbol{s}_0' \cdot c + \boldsymbol{s}_1' = \boldsymbol{s}_0 \cdot c + \boldsymbol{s}_1$ is at least $1 - 2^{-3\lambda-\delta}$. By union bounding over all possible hash values $c \in \mathcal{T}_{\beta_s}$ and observing that $\mathcal{T}_{\beta_s} \leq 2^{2\lambda+\delta}$ the lemma statement follows. □

Lemma 19. *Let $n, \gamma, q, \beta_s, \alpha$ be positive integers with q prime and n a power of two such that $q > 16\alpha\beta_s$ and let $\mathcal{R} = \mathbb{Z}[x]/(x^n + 1)$. Let $\boldsymbol{a} \in \mathcal{R}_q^\gamma$, $c_0, c_1 \in \mathcal{T}_{\beta_s}$, $w_0, w_1 \in \mathcal{T}_\alpha$, and $\sigma_0, \sigma_1 \in \mathcal{R}$ be arbitrary ring elements such that $c_0 \neq c_1$ and $w_0 \neq w_1$. Then there exists at most a single pair of vectors $(\boldsymbol{s}_0, \boldsymbol{s}_1) \in \mathcal{B}_1^\gamma \times \mathcal{B}_{\beta_s}^\gamma$, such that*

$$\boldsymbol{s}_0 \cdot c_0 + \boldsymbol{s}_1 = \sigma_0 \quad \textit{and} \quad (w_0 - w_1) \cdot (\boldsymbol{s}_0 \cdot c_1 + \boldsymbol{s}_1) = \sigma_1.$$

Proof. Let $(\boldsymbol{s}_0, \boldsymbol{s}_1) \in \mathcal{B}_1^\gamma \times \mathcal{B}_{\beta_s}^\gamma$ and $(\boldsymbol{s}'_0, \boldsymbol{s}'_1) \in \mathcal{B}_1^\gamma \times \mathcal{B}_{\beta_s}^\gamma$ be two secret keys, such that

$$\boldsymbol{s}_0 \cdot c_0 + \boldsymbol{s}_1 = \boldsymbol{s}'_0 \cdot c_0 + \boldsymbol{s}'_1 \implies (\boldsymbol{s}_0 - \boldsymbol{s}'_0) \cdot c_0 + (\boldsymbol{s}_1 - \boldsymbol{s}'_1) = 0 \tag{7}$$

and

$$\begin{aligned} &(w_0 - w_1) \cdot (\boldsymbol{s}_0 \cdot c_1 + \boldsymbol{s}_1) = (w_0 - w_1) \cdot (\boldsymbol{s}'_0 \cdot c_1 + \boldsymbol{s}'_1) \\ \implies &(w_0 - w_1)((\boldsymbol{s}_0 - \boldsymbol{s}'_0) \cdot c_1 + (\boldsymbol{s}_1 - \boldsymbol{s}'_1)) = 0 \end{aligned} \tag{8}$$

Equation 7 implies that

$$(w_0 - w_1)((\boldsymbol{s}_0 - \boldsymbol{s}'_0) \cdot c_0 + (\boldsymbol{s}_1 - \boldsymbol{s}'_1)) = 0.$$

Combined with Equation 8, we get that in $\mathcal{R}_q$

$$(w_0 - w_1)(\boldsymbol{s}_0 - \boldsymbol{s}'_0)(c_0 - c_1) = 0 \tag{9}$$

Since $w_0, w_1 \in \mathcal{T}_\alpha$, $\boldsymbol{s}_0, \boldsymbol{s}'_0 \in \mathcal{B}_1^\gamma$, and $c_0, c_1 \in \mathcal{T}_{\beta_s}$, it holds by Lemma 1 that

$$\left\|(w_0 - w_1)(\boldsymbol{s}_0 - \boldsymbol{s}'_0)(c_0 - c_1)\right\| \leq \|w_0 - w_1\|_1 \cdot \|c_0 - c_1\|_1 \cdot \left\|(\boldsymbol{s}_0 - \boldsymbol{s}'_0)\right\| \leq 8\alpha\beta_s \leq \frac{q-1}{2}.$$

Therefore Equation 9 also holds in $\mathcal{R}$. Since $w_0 \neq w_1$, $c_0 \neq c_1$, and $\mathcal{R}$ is an integral domain, it follows that $\boldsymbol{s}_0 = \boldsymbol{s}'_0$. By Equation 7, it must therefore hold that $(\boldsymbol{s}_0, \boldsymbol{s}_1) = (\boldsymbol{s}'_0, \boldsymbol{s}'_1)$. □

5 Synchronized Multi-Signatures

In this section, we present the main construction of this work. Roughly speaking, our construction will produce a public key, which is a vector commitment to a vector of independent one-time signature public keys. To sign a message at time t, the signer will publish an opening to the key in vector position t and then sign the corresponding message with that key. The (non-interactive) aggregation of multiple independent signatures for the same message, will heavily rely on the homomorphic properties of the used vector commitment and one-time signature scheme. Let us now first formally define what a synchronized multi-signature scheme is.

Definition 11 (Synchronized Multi-Signatures). *A synchronized ρ-wise multi-signature scheme for a bounded number of time periods is defined by five PPT algorithms* $(\mathsf{Setup}, \mathsf{KGen}, \mathsf{Sign}, \mathsf{Aggregate}, \mathsf{Vrfy})$.

$\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda, 1^\tau)$ *The setup algorithm takes as input the security parameter and the maximum number of time periods and outputs public parameters* pp.

$(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{KGen}(\mathsf{pp})$ *The key generation algorithm takes as input the public parameters and outputs a key-pair.*

$\sigma \leftarrow \mathsf{Sign}(\mathsf{pp}, \mathsf{sk}, t, m)$ *The signing algorithm takes as input the public parameters, a secret key, a time period $t \in [\tau - 1]$, and a message and outputs a signature.*

$\sigma_{\mathsf{agg}} \leftarrow \mathsf{Aggregate}(\mathsf{pp}, \mathcal{P}, t, m, \mathcal{S})$ *The deterministic aggregation algorithm takes as input the public parameters, a list of public keys, a time period $t \in [\tau - 1]$, a message, and a list of signatures, where $|\mathcal{P}| = |\mathcal{S}| \leq \rho$ and outputs an aggregated signature or an error $\perp$.*

$b \leftarrow \mathsf{Vrfy}(\mathsf{pp}, \mathcal{P}, t, m, \sigma_{\mathsf{agg}})$ *The deterministic verification algorithm takes as input the public parameters, a list of public keys, a time period $t \in [\tau - 1]$, a message, and an aggregated signature and outputs a bit indicating acceptance/rejection.*

A synchronized ρ-wise multi-signature scheme is correct, if for all $\lambda, \tau \in \mathbb{N}$, $\ell \in [\rho] \setminus \{0\}$, $t \in [\tau - 1]$, and $m \in \{0,1\}^*$ it holds that

$$\Pr\left[\begin{array}{c} \mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda, 1^\tau), \\ (\mathsf{sk}_i, \mathsf{pk}_i) \leftarrow \mathsf{KGen}(\mathsf{pp}), \mathcal{P} := (\mathsf{pk}_0, \ldots, \mathsf{pk}_{\ell-1}) \\ \sigma_i \leftarrow \mathsf{Sign}(\mathsf{pp}, \mathsf{sk}_i, t, m), \mathcal{S} := (\sigma_0, \ldots, \sigma_{\ell-1}) \\ \sigma_{\mathsf{agg}} \leftarrow \mathsf{Aggregate}(\mathsf{pp}, \mathcal{P}, t, m, \mathcal{S}) \end{array} : \mathsf{Vrfy}(\mathsf{pp}, \mathcal{P}, t, m, \sigma_{\mathsf{agg}}) = 1\right] = 1$$

Our notion of unforgeability allows for including signatures under adversarially chosen keys into the aggregate signature.

Definition 12 (Unforgeability). *A synchronized ρ-wise multi-signature scheme is unforgeable if for all $\lambda, \tau \in \mathbb{N}$, and all PPT algorithms $\mathcal{A}$ it holds that*

$$\Pr\left[\begin{array}{cc} \mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda, 1^\tau), & \mathsf{Vrfy}(\mathsf{pp}, \mathcal{P}, t, m, \sigma_{\mathsf{agg}}) = 1 \\ (\mathsf{sk}^*, \mathsf{pk}^*) \leftarrow \mathsf{KGen}(\mathsf{pp}) & : \wedge \mathsf{pk}^* \in \mathcal{P} \wedge \nexists\, \sigma\ .\ (t, m, \sigma) \in \mathcal{Q} \\ (\mathcal{P}, t, m, \sigma_{\mathsf{agg}}) \leftarrow \mathcal{A}^{\mathsf{Sign}(\mathsf{pp}, \mathsf{sk}^*, \cdot, \cdot)}(\mathsf{pp}, \mathsf{pk}^*) & \wedge \forall t'.\ |\mathcal{Q}_{t'}| \leq 1 \end{array}\right] \leq \mathsf{negl}(\lambda)$$

for some negligible function $\mathsf{negl}(\lambda)$, where $\mathcal{Q}$ denotes the set of signing queries made by $\mathcal{A}$ and $\mathcal{Q}_{t'}$ denotes the set of signing queries made for timeslot t'.

The following lemma will be useful for proving the security of our construction in Theorem 21, specifically it will be useful during the security reduction to the underlying one-time signature scheme. Intuitively, the lemma shows that two valid aggregate signatures that are created using vectors of random weights that differ in one position, allow for extracting a valid one-time signature and key.

Lemma 20. *Let $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda, 1^\tau)$ and $(\mathsf{sk}^*, \mathsf{pk}^* = c^*) \leftarrow \mathsf{KGen}(\mathsf{pp})$ be fixed. Let $\ell \in [\rho] \setminus \{0\}$, $t \in [\tau - 1]$, $m \in \{0,1\}^*$, $\mathcal{P} = (\mathsf{pk}_0, \ldots, \mathsf{pk}_{\ell-1})$ with $\mathsf{pk}_j = \mathsf{pk}^*$, $\sigma^0_{\mathsf{agg}} = (\sigma'_0, d_0)$, $\sigma^1_{\mathsf{agg}} = (\sigma'_1, d_1)$, and let H_0, H_1 be two random oracles, such that*

$$(w_0, \ldots, w_{\ell-1}) := H_0(t, m, \mathcal{P})$$
$$(w_0, \ldots, w_{j-1}, w'_j, w_{j+1}, \ldots, w_{\ell-1}) := H_1(t, m, \mathcal{P})$$

with $w_j \neq w'_j$ and

$$\mathsf{Vrfy}^{H_0}(\mathsf{pp}, \mathcal{P}, t, m, \sigma^0_{\mathsf{agg}}) = 1 \quad \textit{and} \quad \mathsf{Vrfy}^{H_1}(\mathsf{pp}, \mathcal{P}, t, m, \sigma^1_{\mathsf{agg}}) = 1.$$

Then, for $\mathsf{opk}^* \leftarrow \mathsf{HVC.wVrfy}(\mathsf{pp}_{\mathsf{HVC}}, c^* \cdot (w_j - w'_j), t, d_0 - d_1)$ it holds that

$$\mathsf{opk}^* \neq \perp \quad \textit{and} \quad \mathsf{KOTS.wVrfy}(\mathsf{pp}_{\mathsf{KOTS}}, \mathsf{opk}^*, m, \sigma'_0 - \sigma'_1) = 1.$$

$\mathsf{Setup}(1^\lambda, \tau)$

$\mathsf{pp}_{\mathsf{KOTS}} \leftarrow \mathsf{KOTS.Setup}(1^\lambda)$
$\mathsf{pp}_{\mathsf{HVC}} \leftarrow \mathsf{HVC.Setup}(1^\lambda, \tau)$
return $\mathsf{pp} := (\mathsf{pp}_{\mathsf{KOTS}}, \mathsf{pp}_{\mathsf{HVC}}, \tau)$

$\mathsf{KGen}(\mathsf{pp})$

parse pp **as** $(\mathsf{pp}_{\mathsf{KOTS}}, \mathsf{pp}_{\mathsf{HVC}}, \tau)$
foreach $i \in [2^\tau - 1]$
  $(\mathsf{osk}_i, \mathsf{opk}_i) \leftarrow \mathsf{KOTS.KGen}(\mathsf{pp}_{\mathsf{KOTS}})$
$\mathsf{OSS} = (\mathsf{osk}_0, \ldots, \mathsf{osk}_{2^\tau - 1})$
$\mathsf{OPK} = (\mathsf{opk}_0, \ldots, \mathsf{opk}_{2^\tau - 1})$
$c \leftarrow \mathsf{HVC.Com}(\mathsf{pp}_{\mathsf{HVC}}, \mathsf{OPK})$
return $(\mathsf{sk}, \mathsf{pk}) := ((\mathsf{OSS}, \mathsf{OPK}), c)$

$\mathsf{Aggregate}(\mathsf{pp}, \mathcal{P}, t, m, \mathcal{S})$

parse $\mathcal{S}$ **as** $((\sigma'_0, d_0), \ldots, (\sigma'_{\ell-1}, d_{\ell-1}))$
$(w_0, \ldots, w_{\ell-1}) := H(t, m, \mathcal{P})$
$\sigma' := \sum_{i=0}^{\ell-1} w_i \cdot \sigma'_i$
$d := \sum_{i=0}^{\ell-1} w_i \cdot d_i$
return $\sigma_{\mathsf{agg}} := (\sigma', d)$

$\mathsf{Sign}(\mathsf{pp}, \mathsf{sk}, t, m)$

parse pp **as** $(\mathsf{pp}_{\mathsf{KOTS}}, \mathsf{pp}_{\mathsf{HVC}}, \tau)$
parse sk **as** $((\mathsf{osk}_0, \ldots, \mathsf{osk}_{2^\tau - 1}), \mathsf{OPK})$
$\sigma' \leftarrow \mathsf{KOTS.Sign}(\mathsf{pp}_{\mathsf{KOTS}}, \mathsf{osk}_t, m)$
$d \leftarrow \mathsf{HVC.Open}(\mathsf{pp}_{\mathsf{HVC}}, c, \mathsf{OPK}, t)$
return $\sigma := (\sigma', d)$

$\mathsf{Vrfy}(\mathsf{pp}, \mathcal{P}, t, m, \sigma_{\mathsf{agg}})$

parse pp **as** $(\mathsf{pp}_{\mathsf{KOTS}}, \mathsf{pp}_{\mathsf{HVC}}, \tau)$
parse $\mathcal{P}$ **as** $(c_0, \ldots, c_{\ell-1})$
parse σ_{agg} **as** (σ', d)
if $\ell > \rho$ **or** $t \geq 2^\tau$
  return 0
$(w_0, \ldots, w_{\ell-1}) := H(t, m, \mathcal{P})$
$c := \sum_{i=0}^{\ell-1} w_i \cdot c_i$
$\mathsf{opk} \leftarrow \mathsf{HVC.sVrfy}(\mathsf{pp}_{\mathsf{HVC}}, c, t, d)$
if $\mathsf{opk} = \bot$
  return 0
else
  return $\mathsf{KOTS.sVrfy}(\mathsf{pp}_{\mathsf{KOTS}}, \mathsf{opk}, m, \sigma')$

Fig. 4. The synchronized multi-signature scheme Squirrel based on homomorphic vector commitments and key-homomorphic one-time signatures.

Proof. Since

$$\mathsf{Vrfy}^{H_0}(\mathsf{pp}, \mathcal{P}, t, m, \sigma^0_{\mathsf{agg}}) = 1 \quad \text{and} \quad \mathsf{Vrfy}^{H_1}(\mathsf{pp}, \mathcal{P}, t, m, \sigma^1_{\mathsf{agg}}) = 1,$$

it must hold by definition of the verification algorithm that

$$\mathsf{HVC.sVrfy}(\mathsf{pp}_{\mathsf{HVC}}, \sum_{i \in [\ell-1]} w_i \cdot c_i, t, d_0) = \mathsf{opk}_0$$

and

$$\mathsf{HVC.sVrfy}(\mathsf{pp}_{\mathsf{HVC}}, w'_j \cdot c_j + \sum_{i \in [\ell-1] \setminus \{j\}} w_i \cdot c_i, t, d_1) = \mathsf{opk}_1$$

for $\mathsf{opk}_0, \mathsf{opk}_1 \neq \bot$. Thus by Definition 4 it holds that

$$\begin{aligned}
\mathsf{opk}^* =& \mathsf{HVC.wVrfy}(\mathsf{pp}_{\mathsf{HVC}}, c^* \cdot (w_j - w'_j), t, d_0 - d_1) \\
=& \mathsf{HVC.wVrfy}(\mathsf{pp}_{\mathsf{HVC}}, \Big(\sum_{i \in [\ell-1]} w_i \cdot c_i\Big) - \Big(w'_j \cdot c_j + \sum_{i \in [\ell-1] \setminus \{j\}} w_i \cdot c_i\Big), t, d_0 - d_1) \\
=& (\mathsf{opk}_0 - \mathsf{opk}_1).
\end{aligned}$$

Further, by definition of the verification algorithm it must also hold that

$$\mathsf{KOTS.sVrfy}(\mathsf{pp}_{\mathsf{KOTS}}, \mathsf{opk}_0, m, \sigma'_0) = 1 \quad \text{and} \quad \mathsf{KOTS.sVrfy}(\mathsf{pp}_{\mathsf{KOTS}}, \mathsf{opk}_1, m, \sigma'_1) = 1$$

Thus, by definition Definition 9 it holds that

$$\begin{aligned} &\mathsf{KOTS.wVrfy}(\mathsf{pp}_{\mathsf{KOTS}}, \mathsf{opk}^*, m, \sigma'_0 - \sigma'_1) \\ =&\mathsf{KOTS.wVrfy}(\mathsf{pp}_{\mathsf{KOTS}}, \mathsf{opk}_0 - \mathsf{opk}_1, m, \sigma'_0 - \sigma'_1) = 1 \end{aligned}$$ □

The following theorem now states the security of our construction presented in Figure 4 under the Ring-SIS assumption. The proof of the theorem is relatively long and technical. It essentially works by applying the forking lemma to extract two different aggregated signatures on which Lemma 20 can then be applied. The result of that can then be leveraged to attack either the position binding of the homomorphic vector commitment or the unforgeability of the key-homomorphic one-time signature. We stress again that, due to the use of the forking lemma, this proof does *not* apply to quantum adversaries.

Theorem 21. *Let $\mathcal{R}_q = \mathbb{Z}_q[x]/\langle x^n + 1\rangle$ be a polynomial ring parameterized by $n = \mathsf{poly}(\lambda)$ and $q = \mathsf{poly}(\lambda)$. Let $W \subseteq \mathcal{R}_q$ be a set and let $W' := \{w^0 - w^1 | w^0, w^1 \in W\}$. Let KOTS be a (ρ, W')-homomorphically correct one-time signature scheme with public keys in $\mathcal{R}_q^\xi$ and let HVC be a (ρ, W)-homomorphically correct vector commitment for domain $\mathcal{R}_q^\xi$. If KOTS is robustly homomorphic and multi-user existentially unforgeable under rerandomized keys and HVC is robustly homomorphic and position-binding, then Squirrel, shown in Figure 4, is a correct and unforgeable synchronized ρ-wise multi-signature.*

Proof. It is easy to verify, that the homomorphic correctness of the vector commitment and the one-time signature scheme imply that the scheme is correct.

To prove unforgeability, let $\mathcal{A}$ be an arbitrary PPT algorithm that makes at most $p = \mathsf{poly}(\lambda)$ queries to the random oracle and for which it holds that

$$\Pr\left[\begin{array}{rl} \mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda, \rho, \tau); & \mathsf{Vrfy}(\mathsf{pp}, \mathcal{P}^*, t^*, m^*, \sigma^*_{\mathsf{agg}}) = 1 \\ (\mathsf{sk}^*, \mathsf{pk}^*) \leftarrow \mathsf{KGen}(\mathsf{pp}); & : \wedge \mathsf{pk}^* \in \mathcal{P} \wedge \nexists\, \sigma\ .\ (t^*, m^*, \sigma) \in \mathcal{Q} \\ (\mathcal{P}^*, t^*, m^*, \sigma^*_{\mathsf{agg}}) \leftarrow \mathcal{A}^{\mathsf{Sign}(\mathsf{pp}, \mathsf{sk}^*, \cdot, \cdot)}(\mathsf{pp}, \mathsf{pk}^*) & \wedge \forall t'.\ |\mathcal{Q}_{t'}| \leq 1 \end{array}\right] = \epsilon(\lambda) \quad (10)$$

We assume without loss of generality that $\mathcal{A}$ always queries $(\mathcal{P}^*, t^*, m^*)$ to the random oracle.

Claim 22. *Let IG and $\mathcal{B}$ be as defined in Figure 5. Then it holds that*

$$\Pr\left[\begin{array}{c} ((\mathsf{pp}, \mathsf{pk}^*), (\mathsf{pp}, \mathsf{sk}^*)) \leftarrow \mathsf{IG}(1^\lambda); \\ h_0, \ldots, h_{p-1} \leftarrow W; : b = 1 \\ (b, i, \omega) \leftarrow \mathcal{B}^{\mathsf{Sign}(\mathsf{pp}, \cdot)}(x, h_0, \ldots, h_{p-1}) \end{array}\right] = \epsilon(\lambda).$$

Proof. The input generation algorithm IG performs exactly the same setup expected by $\mathcal{A}$ and then $\mathcal{B}$ simply executes $\mathcal{A}$, perfectly simulating the random oracle by lazy sampling. The only interesting part of the simulation is the fact that $\mathcal{B}$ reorders the used randomness if a query includes the challenge public key pk^*. This is necessary to later make use of Lemma 2 without modification, but does not impact the simulation at all: the random values h_0 through $h_{p\cdot\rho-1}$ are all distributed

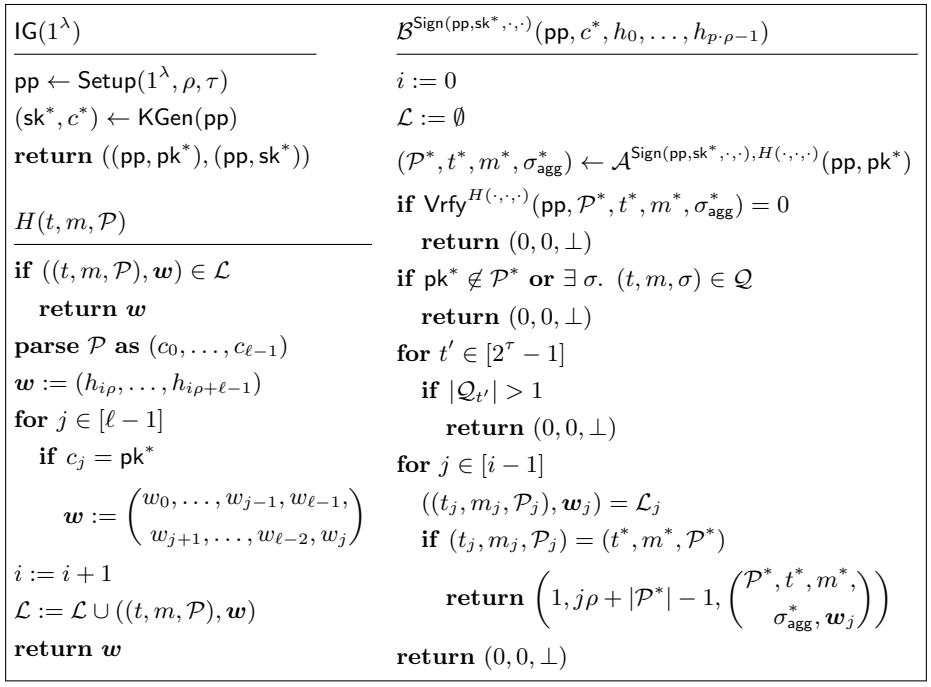
$\mathsf{IG}(1^\lambda)$
$\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda, \rho, \tau)$
$(\mathsf{sk}^*, c^*) \leftarrow \mathsf{KGen}(\mathsf{pp})$
return $((\mathsf{pp}, \mathsf{pk}^*), (\mathsf{pp}, \mathsf{sk}^*))$
$H(t, m, \mathcal{P})$
if $((t, m, \mathcal{P}), \boldsymbol{w}) \in \mathcal{L}$
return $\boldsymbol{w}$
parse $\mathcal{P}$ **as** $(c_0, \ldots, c_{\ell-1})$
$\boldsymbol{w} := (h_{i\rho}, \ldots, h_{i\rho+\ell-1})$
for $j \in [\ell - 1]$
if $c_j = \mathsf{pk}^*$
$\boldsymbol{w} := \begin{pmatrix} w_0, \ldots, w_{j-1}, w_{\ell-1}, \\ w_{j+1}, \ldots, w_{\ell-2}, w_j \end{pmatrix}$
$i := i + 1$
$\mathcal{L} := \mathcal{L} \cup ((t, m, \mathcal{P}), \boldsymbol{w})$
return $\boldsymbol{w}$
$\mathcal{B}^{\mathsf{Sign}(\mathsf{pp},\mathsf{sk}^*,\cdot,\cdot)}(\mathsf{pp}, c^*, h_0, \ldots, h_{p\cdot\rho-1})$
$i := 0$
$\mathcal{L} := \emptyset$
$(\mathcal{P}^*, t^*, m^*, \sigma^*_{\mathsf{agg}}) \leftarrow \mathcal{A}^{\mathsf{Sign}(\mathsf{pp},\mathsf{sk}^*,\cdot,\cdot),H(\cdot,\cdot,\cdot)}(\mathsf{pp}, \mathsf{pk}^*)$
if $\mathsf{Vrfy}^{H(\cdot,\cdot,\cdot)}(\mathsf{pp}, \mathcal{P}^*, t^*, m^*, \sigma^*_{\mathsf{agg}}) = 0$
return $(0, 0, \bot)$
if $\mathsf{pk}^* \notin \mathcal{P}^*$ **or** $\exists\, \sigma.\ (t, m, \sigma) \in \mathcal{Q}$
return $(0, 0, \bot)$
for $t' \in [2^\tau - 1]$
if $|\mathcal{Q}_{t'}| > 1$
return $(0, 0, \bot)$
for $j \in [i - 1]$
$((t_j, m_j, \mathcal{P}_j), \boldsymbol{w}_j) = \mathcal{L}_j$
if $(t_j, m_j, \mathcal{P}_j) = (t^*, m^*, \mathcal{P}^*)$
return $\left(1, j\rho + |\mathcal{P}^*| - 1, \begin{pmatrix} \mathcal{P}^*, t^*, m^*, \\ \sigma^*_{\mathsf{agg}}, \boldsymbol{w}_j \end{pmatrix}\right)$
return $(0, 0, \bot)$

Fig. 5. The setup for the forking lemma based on attacker $\mathcal{A}$.

independently and the swapping takes place independently of their values. Therefore the distribution of the random oracle answers is identical with or without swapping. After executing $\mathcal{A}$, the algorithm $\mathcal{B}$ checks whether $\mathcal{A}$ would have been successful according to Definition 12 and outputs $b = 1$ iff $\mathcal{A}$ was successful. Therefore, the claim follows. □

By combining Claim 22 with Lemma 2 we can conclude that

$$\epsilon \leq \frac{p}{|\mathcal{T}_\alpha|} + \sqrt{p \cdot \Pr\left[\begin{matrix} ((\mathsf{pp}, \mathsf{pk}^*), (\mathsf{pp}, \mathsf{sk}^*)) \leftarrow \mathsf{IG}(1^\lambda); \\ (b, \omega^0, \omega^1) \leftarrow \mathsf{F}_{\mathcal{B}}^{\mathsf{Sign}(\mathsf{pp},\mathsf{sk}^*,\cdot,\cdot)}(\mathsf{pp}, \mathsf{pk}^*) \end{matrix} : b = 1\right]}. \tag{11}$$

It remains to bound the probability

$$\Pr\left[\begin{matrix} ((\mathsf{pp}, \mathsf{pk}^*), (\mathsf{pp}, \mathsf{sk}^*)) \leftarrow \mathsf{IG}(1^\lambda); \\ (b, \omega^0, \omega^1) \leftarrow \mathsf{F}_{\mathcal{B}}^{\mathsf{Sign}(\mathsf{pp},\mathsf{sk}^*,\cdot,\cdot)}(\mathsf{pp}, \mathsf{pk}^*) \end{matrix} : b = 1\right]$$

For any output $(1, (\mathcal{P}^0, t^0, m^0, (\sigma^0, d^0), \boldsymbol{w}^0), (\mathcal{P}^1, t^1, m^1, (\sigma^1, d^1), \boldsymbol{w}^1))$ of the forking algorithm it holds by definition of $\mathcal{B}$ and $\mathsf{F}_{\mathcal{B}}$ that $(\mathcal{P}^0, t^0, m^0) = (\mathcal{P}^1, t^1, m^1)$ because these are inputs to the random oracle *before* the fork occurs. To improve readability we thus introduce a modified forking algorithm $\widetilde{\mathsf{F}}_{\mathcal{B}}$ defined in Figure 6. Obviously, it holds that

$$\begin{aligned} &\Pr\left[\begin{matrix} ((\mathsf{pp}, \mathsf{pk}^*), (\mathsf{pp}, \mathsf{sk}^*)) \leftarrow \mathsf{IG}(1^\lambda); \\ (b, \omega^0, \omega^1) \leftarrow \mathsf{F}_{\mathcal{B}}^{\mathsf{Sign}(\mathsf{pp},\mathsf{sk}^*,\cdot,\cdot)}(\mathsf{pp}, \mathsf{pk}^*) \end{matrix} : b = 1\right] \\ =&\Pr\left[\begin{matrix} ((\mathsf{pp}, \mathsf{pk}^*), (\mathsf{pp}, \mathsf{sk}^*)) \leftarrow \mathsf{IG}(1^\lambda); \\ (b, \mathcal{P}, t, m, \sigma^0_{\mathsf{agg}}, \boldsymbol{w}^0, \sigma^1_{\mathsf{agg}}, \boldsymbol{w}^1) \leftarrow \widetilde{\mathsf{F}}_{\mathcal{B}}^{\mathsf{Sign}(\mathsf{pp},\mathsf{sk}^*,\cdot,\cdot)}(\mathsf{pp}, \mathsf{pk}^*) \end{matrix} : b = 1\right] \end{aligned} \tag{12}$$

$\widetilde{\mathsf{F}}_{\mathcal{B}}^{\mathsf{Sign}(\mathsf{pp},\mathsf{sk}^*)}(\mathsf{pp},\mathsf{pk}^*)$

$(b,(\mathcal{P}^0,t^0,m^0,\sigma^0_{\mathsf{agg}},\boldsymbol{w}^0),(\mathcal{P}^1,t^1,m^1,\sigma^1_{\mathsf{agg}},\boldsymbol{w}^1)) \leftarrow \mathsf{F}_{\mathcal{B}}^{\mathsf{Sign}(\mathsf{pp},\mathsf{sk}^*,\cdot,\cdot)}(\mathsf{pp},\mathsf{pk}^*)$

return $(b,\mathcal{P}^0,t^0,m^0,\sigma^0_{\mathsf{agg}},\boldsymbol{w}^0,\sigma^1_{\mathsf{agg}},\boldsymbol{w}^1)$

Fig. 6. The modified forking algorithm.

Let $(b,\mathcal{P},t,m,(\sigma^0,d^0),\boldsymbol{w}^0,(\sigma^1,d^1),\boldsymbol{w}^1) \leftarrow \widetilde{\mathsf{F}}_{\mathcal{B}}^{\mathsf{Sign}(\mathsf{pp},\mathsf{sk}^*,\cdot,\cdot)}(\mathsf{pp},\mathsf{pk}^*)$ be an execution of the modified forking algorithm with

$$\mathsf{pk}^* = c^* \quad \text{and} \quad \mathsf{sk}^* = ((\mathsf{osk}^*_0,\ldots,\mathsf{osk}^*_{2^\tau-1}),(\mathsf{opk}^*_0,\ldots,\mathsf{opk}^*_{2^\tau-1})).$$

Let j denote the index, such that $\mathcal{P}_j = \mathsf{pk}^*$. We then define

$$\widetilde{w} := (w^0_j - w^1_j) \quad \text{and} \quad \widetilde{\mathsf{opk}} := \mathsf{HVC.wVrfy}(\mathsf{pp}_{\mathsf{HVC}},\widetilde{w}\cdot c^*,t,(d^0-d^1)).$$

The probability from Equation 12 can then be split as

$$\begin{aligned}
&\Pr\left[\begin{array}{c}((\mathsf{pp},\mathsf{pk}^*),(\mathsf{pp},\mathsf{sk}^*)) \leftarrow \mathsf{IG}(1^\lambda);\\ (b,\mathcal{P},t,m,\sigma^0_{\mathsf{agg}},\boldsymbol{w}^0,\sigma^1_{\mathsf{agg}},\boldsymbol{w}^1) \leftarrow \widetilde{\mathsf{F}}_{\mathcal{B}}^{\mathsf{Sign}(\mathsf{pp},\mathsf{sk}^*,\cdot,\cdot)}(\mathsf{pp},\mathsf{pk}^*)\end{array} : b=1\right]\\
=&\Pr\left[\begin{array}{c}((\mathsf{pp},\mathsf{pk}^*),(\mathsf{pp},\mathsf{sk}^*)) \leftarrow \mathsf{IG}(1^\lambda);\\ (b,\mathcal{P},t,m,\sigma^0_{\mathsf{agg}},\boldsymbol{w}^0,\sigma^1_{\mathsf{agg}},\boldsymbol{w}^1) \leftarrow \widetilde{\mathsf{F}}_{\mathcal{B}}^{\mathsf{Sign}(\mathsf{pp},\mathsf{sk}^*,\cdot,\cdot)}(\mathsf{pp},\mathsf{pk}^*)\end{array} : \begin{array}{c}\widetilde{\mathsf{opk}} = \widetilde{w}\cdot\mathsf{opk}^*_t\\ \wedge b=1\end{array}\right] \qquad (13)\\
&+\Pr\left[\begin{array}{c}((\mathsf{pp},\mathsf{pk}^*),(\mathsf{pp},\mathsf{sk}^*)) \leftarrow \mathsf{IG}(1^\lambda);\\ (b,\mathcal{P},t,m,\sigma^0_{\mathsf{agg}},\boldsymbol{w}^0,\sigma^1_{\mathsf{agg}},\boldsymbol{w}^1) \leftarrow \widetilde{\mathsf{F}}_{\mathcal{B}}^{\mathsf{Sign}(\mathsf{pp},\mathsf{sk}^*,\cdot,\cdot)}(\mathsf{pp},\mathsf{pk}^*)\end{array} : \begin{array}{c}\widetilde{\mathsf{opk}} \neq \widetilde{w}\cdot\mathsf{opk}^*_t\\ \wedge b=1\end{array}\right]
\end{aligned}$$

and we can bound the two parts separately.

Consider $\mathcal{R}_0$ described in Figure 7 as an adversary against the position binding of HVC. To analyse the success probability of $\mathcal{R}_0$, consider an execution $(c,t,d_0,d_1) \leftarrow \mathcal{R}_0(\mathsf{pp}_{\mathsf{HVC}},\tau)$. According to the definitions above, it holds that

$$\begin{aligned}
&\mathsf{HVC.wVrfy}(\mathsf{pp}_{\mathsf{HVC}},c,t,d_0)\\
=&\mathsf{HVC.wVrfy}(\mathsf{pp}_{\mathsf{HVC}},w^0_j\cdot c^* - w^1_j\cdot c^*,t^0,w^0_j\cdot\mathsf{Open}(\mathsf{pp}_{\mathsf{HVC}},c,\mathsf{OPK}^*,t) - w^0_j\cdot\mathsf{Open}(\mathsf{pp}_{\mathsf{HVC}},c,\mathsf{OPK}^*,t))\\
=&\mathsf{HVC.sVrfy}(\mathsf{pp}_{\mathsf{HVC}},w^0_j\cdot c^*,t^0,w^0_j\cdot\mathsf{Open}(\mathsf{pp}_{\mathsf{HVC}},c,\mathsf{OPK}^*,t))\\
&-\mathsf{HVC.sVrfy}(\mathsf{pp}_{\mathsf{HVC}},w^1_j\cdot c^*,t^0,w^1_j\cdot\mathsf{Open}(\mathsf{pp}_{\mathsf{HVC}},c,\mathsf{OPK}^*,t)) && \text{(Robust Homomorphism)}\\
=&w^0_j\cdot\mathsf{opk}^*_t - w^1_j\cdot\mathsf{opk}^*_t = \widetilde{w}\cdot\mathsf{opk}^*_t && \text{(Homomorphic correctness)}
\end{aligned}$$

and

$$\mathsf{HVC.wVrfy}(\mathsf{pp}_{\mathsf{HVC}},c,t,d_1) = \mathsf{HVC.wVrfy}(\mathsf{pp}_{\mathsf{HVC}},\widetilde{w}\cdot c^*,t,(d^0-d^1)) = \widetilde{\mathsf{opk}}$$

We then have that

$$\Pr\left[\begin{array}{r}\mathsf{pp}_{\mathsf{HVC}} \leftarrow \mathsf{HVC.Setup}(1^\lambda,\tau);\\ (c,t,d_0,d_1) \leftarrow \mathcal{R}_0(\mathsf{pp});\\ m_0 \leftarrow \mathsf{HVC.wVrfy}(\mathsf{pp},c,t,d_0);\\ m_1 \leftarrow \mathsf{HVC.wVrfy}(\mathsf{pp},c,t,d_1)\end{array} : m_0 \neq m_1 \wedge \perp \notin \{m_0,m_1\}\right]$$

$\mathcal{R}_0(\mathsf{pp}_{\mathsf{HVC}}, \tau)$

$\mathsf{pp}_{\mathsf{KOTS}} \leftarrow \mathsf{KOTS.Setup}(1^\lambda)$
$\mathsf{pp} := (\mathsf{pp}_{\mathsf{KOTS}}, \mathsf{pp}_{\mathsf{HVC}}, \tau)$
$((\mathsf{OSS}^*, \mathsf{OPK}^*), c^*) \leftarrow \mathsf{KGen}(\mathsf{pp})$
$(b, \mathcal{P}, t, m, \sigma^0_{\mathsf{agg}}, \boldsymbol{w}^0, \sigma^1_{\mathsf{agg}}, \boldsymbol{w}^1) \leftarrow \widetilde{\mathsf{F}}_{\mathcal{B}}^{\mathsf{Sign}(\mathsf{pp},(\mathsf{OSS}^*,\mathsf{OPK}^*),\cdot,\cdot)}(\mathsf{pp}, c^*)$
if $b = 1$
  parse $\mathcal{P}$ **as** $(c_0, \ldots, c_\ell)$
  parse σ^0_{agg} **as** (σ^0, d^0)
  parse σ^1_{agg} **as** (σ^1, d^1)
  for $j \in [\ell - 1]$
    if $c_j = c^*$
      return $(\widetilde{w} \cdot c^*, t, \widetilde{w} \cdot \mathsf{Open}(\mathsf{pp}_{\mathsf{HVC}}, c^*, \mathsf{OPK}^*, t), (d^0 - d^1))$
return $\perp$

Fig. 7. A reduction that uses the forking algorithm $\widetilde{\mathsf{F}}_{\mathcal{B}}$ to attack the position binding of HVC.

$$= \Pr\left[\begin{array}{r} ((\mathsf{pp}, \mathsf{pk}^*), (\mathsf{pp}, \mathsf{sk}^*)) \leftarrow \mathsf{IG}(1^\lambda); \\ (b, \mathcal{P}, t, m, \sigma^0_{\mathsf{agg}}, \boldsymbol{w}^0, \sigma^1_{\mathsf{agg}}, \boldsymbol{w}^1) \leftarrow \widetilde{\mathsf{F}}_{\mathcal{B}}^{\mathsf{Sign}(\mathsf{pp},\mathsf{sk}^*,\cdot,\cdot)}(\mathsf{pp}, \mathsf{pk}^*); \\ m_0 := \widetilde{w} \cdot \mathsf{opk}^*_t; m_1 := \widetilde{\mathsf{opk}} \end{array} : \begin{array}{c} m_0 \neq m_1 \\ \wedge \perp \notin \{m_0, m_1\} \end{array}\right] \tag{14}$$

$$= \Pr\left[\begin{array}{r} ((\mathsf{pp}, \mathsf{pk}^*), (\mathsf{pp}, \mathsf{sk}^*)) \leftarrow \mathsf{IG}(1^\lambda); \\ (b, \mathcal{P}, t, m, \sigma^0_{\mathsf{agg}}, \boldsymbol{w}^0, \sigma^1_{\mathsf{agg}}, \boldsymbol{w}^1) \leftarrow \widetilde{\mathsf{F}}_{\mathcal{B}}^{\mathsf{Sign}(\mathsf{pp},\mathsf{sk}^*,\cdot,\cdot)}(\mathsf{pp}, \mathsf{pk}^*) \end{array} : \widetilde{\mathsf{opk}} \neq \widetilde{w} \cdot \mathsf{opk}^*_t \wedge \widetilde{\mathsf{opk}} \neq \perp\right] \tag{15}$$

$$\geq \Pr\left[\begin{array}{r} ((\mathsf{pp}, \mathsf{pk}^*), (\mathsf{pp}, \mathsf{sk}^*)) \leftarrow \mathsf{IG}(1^\lambda); \\ (b, \mathcal{P}, t, m, \sigma^0_{\mathsf{agg}}, \boldsymbol{w}^0, \sigma^1_{\mathsf{agg}}, \boldsymbol{w}^1) \leftarrow \widetilde{\mathsf{F}}_{\mathcal{B}}^{\mathsf{Sign}(\mathsf{pp},\mathsf{sk}^*,\cdot,\cdot)}(\mathsf{pp}, \mathsf{pk}^*) \end{array} : \widetilde{\mathsf{opk}} \neq \widetilde{w} \cdot \mathsf{opk}^*_t \wedge b = 1\right] \tag{16}$$

where Equation 14 follows because the inputs $\mathcal{R}_0$ provides to the forking algorithm are distributed identically to those sampled by IG. Equation 15 follows because by the above observation $(w^0_j - w^1_j) \cdot \mathsf{opk}^*_t$ cannot be $\perp$. Finally Equation 16 follows from Lemma 20 and the fact that the forking algorithm only outputs $b = 1$ if both multi-signatures verify.

Since the homomorphic vector commitment is assumed to be position binding, it thus follows that

$$\mathsf{negl}(\lambda) \geq \Pr\left[\begin{array}{r} ((\mathsf{pp}, \mathsf{pk}^*), (\mathsf{pp}, \mathsf{sk}^*)) \leftarrow \mathsf{IG}(1^\lambda); \\ (b, \mathcal{P}, t, m, \sigma^0_{\mathsf{agg}}, \boldsymbol{w}^0, \sigma^1_{\mathsf{agg}}, \boldsymbol{w}^1) \leftarrow \widetilde{\mathsf{F}}_{\mathcal{B}}^{\mathsf{Sign}(\mathsf{pp},\mathsf{sk}^*,\cdot,\cdot)}(\mathsf{pp}, \mathsf{pk}^*) \end{array} : \begin{array}{r} \widetilde{\mathsf{opk}} \neq \widetilde{w} \cdot \mathsf{opk}^*_t \\ \wedge b = 1 \end{array}\right] \tag{17}$$

Consider now $\mathcal{R}_1$ described in Figure 8 as an adversary against the existential unforgeability under rerandomized keys of KOTS. We analyze the success probability of $\mathcal{R}_1$ by observing that

$$\Pr\left[\begin{array}{r} \mathsf{pp}_{\mathsf{KOTS}} \leftarrow \mathsf{KOTS.Setup}(1^\lambda); \\ \forall i \in [T-1]. (\mathsf{osk}_i, \mathsf{opk}_i) \leftarrow \mathsf{KOTS.KGen}(\mathsf{pp}); \\ (i^*, m^*, \sigma^*, \widetilde{w}) \leftarrow \mathcal{R}_1^{\widetilde{\mathsf{Sign}}(\cdot,\cdot)}(\mathsf{pp}_{\mathsf{KOTS}}, \mathsf{pp}, \mathsf{OPK}); \end{array} : \begin{array}{r} \mathsf{KOTS.wVrfy}(\mathsf{pp}, \widetilde{\mathsf{opk}}_{i^*}, m^*, \sigma^*) = 1 \\ \wedge m^* \notin Q_{i^*} \wedge \widetilde{w} \in W' \end{array}\right] \tag{18}$$

$\mathcal{R}_1(\mathsf{pp}_{\mathsf{KOTS}}, \mathsf{opk}_0, \ldots \mathsf{opk}_{2^\tau-1})$

$\mathsf{pp}_{\mathsf{HVC}} \leftarrow \mathsf{HVC.Setup}(1^\lambda)$
$\mathsf{pp} := (\mathsf{pp}_{\mathsf{KOTS}}, \mathsf{pp}_{\mathsf{HVC}}, \tau)$
$\mathsf{OPK}^* := (\mathsf{opk}_0, \ldots, \mathsf{opk}_{2^\tau-1})$
$c^* \leftarrow \mathsf{HVC.Com}(\mathsf{pp}_{\mathsf{HVC}}, \mathsf{OPK}^*)$
$(b, \mathcal{P}, t, m, \sigma^0_{\mathsf{agg}}, \boldsymbol{w}^0, \sigma^1_{\mathsf{agg}}, \boldsymbol{w}^1) \leftarrow \widetilde{\mathsf{F}}^{\widetilde{\mathsf{Sign}}(\cdot,\cdot)}_{\mathcal{B}}(\mathsf{pp}, c^*)$
if $b = 1$
  parse $\mathcal{P}$ **as** $(c_0, \ldots, c_\ell)$
  parse σ^0_{agg} **as** (σ^0, d^0)
  parse σ^1_{agg} **as** (σ^1, d^1)
  for $j \in [\ell - 1]$
    if $c_j = c^*$
      return $(t, m, (\sigma^0 - \sigma^1), w^0_j - w^1_j)$
return $\perp$

Fig. 8. A reduction that uses the forking algorithm $\widetilde{\mathsf{F}}_{\mathcal{B}}$ to attack the multi-user existential unforgeability under rerandomized keys of KOTS.

$$\geq \Pr\left[\begin{array}{r} ((\mathsf{pp}, \mathsf{pk}^*), (\mathsf{pp}, \mathsf{sk}^*)) \leftarrow \mathsf{IG}(1^\lambda); \\ \begin{pmatrix} b, \mathcal{P}, t, m, \\ \sigma^0_{\mathsf{agg}}, \boldsymbol{w}^0, \\ \sigma^1_{\mathsf{agg}}, \boldsymbol{w}^1 \end{pmatrix} \leftarrow \widetilde{\mathsf{F}}^{\mathsf{Sign}(\mathsf{pp},\mathsf{sk}^*,\cdot,\cdot)}_{\mathcal{B}}(\mathsf{pp}, \mathsf{pk}^*) : \begin{array}{r} \mathsf{KOTS.wVrfy}(\mathsf{pp}, \widetilde{w} \cdot \mathsf{opk}_t, m, \sigma^0 - \sigma^1) = 1 \\ \wedge (\nexists \sigma'.\, (m, t, \sigma') \in Q) \end{array} \end{array}\right] \tag{19}$$

$$\geq \Pr\left[\begin{array}{r} ((\mathsf{pp}, \mathsf{pk}^*), (\mathsf{pp}, \mathsf{sk}^*)) \leftarrow \mathsf{IG}(1^\lambda); \\ \begin{pmatrix} b, \mathcal{P}, t, m, \\ \sigma^0_{\mathsf{agg}}, \boldsymbol{w}^0, \\ \sigma^1_{\mathsf{agg}}, \boldsymbol{w}^1 \end{pmatrix} \leftarrow \widetilde{\mathsf{F}}^{\mathsf{Sign}(\mathsf{pp},\mathsf{sk}^*,\cdot,\cdot)}_{\mathcal{B}}(\mathsf{pp}, \mathsf{pk}^*) : \begin{array}{r} \widetilde{\mathsf{opk}} = \widetilde{w} \cdot \mathsf{opk}_t \\ \wedge \mathsf{KOTS.wVrfy}(\mathsf{pp}, \widetilde{\mathsf{opk}}, m, \sigma^0 - \sigma^1) = 1 \\ \wedge (\nexists \sigma'.\, (m, t, \sigma') \in Q) \end{array} \end{array}\right] \tag{20}$$

$$\geq \Pr\left[\begin{array}{r} ((\mathsf{pp}, \mathsf{pk}^*), (\mathsf{pp}, \mathsf{sk}^*)) \leftarrow \mathsf{IG}(1^\lambda); \\ \begin{pmatrix} b, \mathcal{P}, t, m, \\ \sigma^0_{\mathsf{agg}}, \boldsymbol{w}^0, \\ \sigma^1_{\mathsf{agg}}, \boldsymbol{w}^1 \end{pmatrix} \leftarrow \widetilde{\mathsf{F}}^{\mathsf{Sign}(\mathsf{pp},\mathsf{sk}^*,\cdot,\cdot)}_{\mathcal{B}}(\mathsf{pp}, \mathsf{pk}^*) : \begin{array}{r} \widetilde{\mathsf{opk}} = \widetilde{w} \cdot \mathsf{opk}_{t^*} \\ \wedge b = 1 \end{array} \end{array}\right]. \tag{21}$$

Here Equation 19 follows by observing that the inputs $\mathcal{R}_1$ provides to the forking algorithm are distributed identically to those sampled by IG and $\mathcal{R}_1$ is successful whenever the difference of the two signatures verifies. Finally, Equation 21 follows from Lemma 20 and the fact that the forking algorithm only outputs $b = 1$ if both multi-signatures verify.

Since KOTS is assumed to be multi-user existentially unforgeable under rerandomized keys, it thus follows that

$$\mathsf{negl}(\lambda) \geq \Pr\left[\begin{array}{c} ((\mathsf{pp},\mathsf{pk}^*),(\mathsf{pp},\mathsf{sk}^*)) \leftarrow \mathsf{IG}(1^\lambda); \\ \begin{pmatrix} b,\mathcal{P},t,m, \\ \sigma^0_{\mathsf{agg}},\boldsymbol{w}^0, \\ \sigma^1_{\mathsf{agg}},\boldsymbol{w}^1 \end{pmatrix} \leftarrow \widetilde{\mathsf{F}}^{\mathsf{Sign}(\mathsf{pp},\mathsf{sk}^*,\cdot,\cdot)}_{\mathcal{B}}(\mathsf{pp},\mathsf{pk}^*) \end{array} : \begin{array}{r} \widetilde{\mathsf{opk}} = \widetilde{w} \cdot \mathsf{opk}_{t^*} \\ \wedge b = 1 \end{array}\right] \tag{22}$$

Combining Equations 17 and 22 with Equation 11 we can conclude that

$$\epsilon \leq \frac{p}{|\mathcal{T}_\alpha|} + \sqrt{p \cdot \mathsf{negl}(\lambda)}$$

which implies that ϵ is negligible and concludes the proof. □

6 Parameters and performances

Let us first set up our stage with Ethereum blockchain as a running example. It is reported that there are over 300,000 nodes in total [MEJ20], and an Ethereum block is agreed by around 2500 active validators within 10 seconds [Eth22]. We therefore target $\rho = 4096$ signature aggregations which is more than enough to aggregate all the votes from those validators. To illustrate the scalability of our scheme, we also present data for $\rho = 1024$ and 8192, respectively.

Our synchronized multi-signature scheme "Squirrel" uses a time parameter τ, which also defines the height for our labeled binary tree. We give parameters for $\tau \in \{21, 24, 26\}$, that roughly translate to 0.66, 5 and 21 years of life time for public keys, if we assume each block takes 10 seconds to generate.

6.1 Parameters and space complexity

ρ	1024	4096	8192
n	512		
$q_{\mathtt{HVC}}$	12289	61441	249857
$q_{\mathtt{KOTS}}$	6694913	28930049	57673729
α	20		
β_s	44		
β_{agg}	2048	4096	8192
γ	41	44	46

Table 2. Parameter sets

We propose three parameter sets each targeting 112 bits security as in Table 2. For the rest of the section, we will use $\rho = 4096$ as an example. We set $q_{\mathtt{KOTS}} = 28930049$ and $q_{\mathtt{HVC}} = 61441$ respectively; both are NTT friendly for our choice of $n = 512$. This implies that our $\mathsf{bin}(\cdot)$ maps an $\mathcal{R}_{q_{\mathtt{HVC}}}$ element into 16 elements; and maps an $\mathcal{R}_{q_{\mathtt{KOTS}}}$ element into 25 elements. Note that $\mathsf{bin}(\cdot)$ does not map a random elements in $\mathcal{R}_{q_{\mathtt{HVC}}}$ uniformly to binary polynomials space; neither does our scheme require such a uniformity. It takes $n\lceil \log q_{\mathtt{HVC}} \rceil = 8192$ bits, or 1 kilobyte to represent an element in $\mathcal{R}_{q_{\mathtt{HVC}}}$. A Squirrel signature consists of three components:

– an HVC decommitment of 2τ path nodes and adjacent nodes, where each node consists of $\lceil \log q_{\mathtt{HVC}} \rceil$ many $\mathcal{R}_{q_{\mathtt{HVC}}}$ elements with bounded norm $\beta_{\mathtt{agg}}$;
– a KOTS public key and its sibling public key, which are hashed into the committed leaves. This consists of $4\lceil \log q_{\mathtt{KOTS}} \rceil$ many $\mathcal{R}_{q_{\mathtt{HVC}}}$ elements with bounded norm $\beta_{\mathtt{agg}}$;
– a KOTS signature, that consists of γ many $\mathcal{R}_{q_{\mathtt{KOTS}}}$ elements with bounded norm β_σ.

For a fresh signature (prior to aggregation), the polynomials in each node are all binary, derived from a decomposition of a single $\mathcal{R}_{q_{\mathtt{HVC}}}$ element. It is therefore sufficient to represent the node with 1 kilobytes of data. In addition, since the signature has not been aggregated, one will be able to derive the nodes along the path with the adjacent leaf and τ adjacent nodes. This reduces the required number of nodes to τ, excluding the root (a.k.a. the public key). In total, we require τ kilobytes storage for a path when the signature is not aggregated.

During aggregation, we multiply the binary polynomials from different users but at a same position from the tree with randomizers, and sum up the products. This gives us a total number of $2\tau\lceil \log q_{\mathtt{HVC}} \rceil$ polynomials, where each polynomial has an infinity norm bound $\alpha\rho$. In practice, it is possible to derive a better bound $\beta_{\mathtt{agg}} = 4096$ if we assume that the polynomials in the decommitments are all binary. We defer this discuss to Section 6.3. An aggregated path requires a maximum $2\tau\lceil \log q_{\mathtt{HVC}} \rceil n(\log \beta_{\mathtt{agg}} + 1)$ bits, or 26τ KB of data.

For the KOTS, prior to aggregation, each public key consists of 2 $\mathcal{R}_{q_{\mathtt{KOTS}}}$ elements, of a combined size of 3.1 KB. During aggregation, each public key is decomposed into $\lceil \log q_{\mathtt{KOTS}} \rceil$ many $\mathcal{R}_{q_{\mathtt{HVC}}}$ elements. The aggregated polynomials also have a same norm bound of $\beta_{\mathtt{agg}}$. That is, an aggregated KOTS public key requires a maximum $2\lceil \log q_{\mathtt{KOTS}} \rceil n(\log \beta_{\mathtt{agg}} + 1)$ bits, or 40.6 KB of data.

A non-aggregated KOTS signature requires γ ring elements with a norm bound of $2\beta_s$, or $n\gamma(\lceil \log(2\beta_s) \rceil + 1) = 22$ KB. We defer to Section 6.3 for how β_s is chosen. An aggregated KOTS signature requires γ ring elements with a norm bound $\beta_\sigma = 2\rho\alpha\beta_s$, which is $n\gamma(\lceil \log(\beta_\sigma) \rceil + 1) = 66$ KB.

Putting everything together, our scheme's public key is the root of the tree that uses 1 kilobytes. An un-aggregated signature requires $\tau + 28$ kilobytes, consists of the path to the root, which is τ nodes; two KOTS public keys of 6.2 kilobytes, and a KOTS signature that requires 22 KB. An aggregated signature requires $26\tau + 147$ kilobytes, consists of the HVC decommitment, which is 2τ number of nodes; two aggregated KOTS public keys of 81.2 KB, and a KOTS signature of 66 KB.

We summarize the characteristics of our scheme in Table 3.

ρ: #sig	τ: tree height	Life cycle	PK size	Sig size	Max AggSig size	Improvement[a]
1024	21	8 months	0.9 KB	45 KB	572 KB	14%
	24	5 years		48 KB	635 KB	5%
	26	21 years		50 KB	677 KB	
4096	21	8 months	1 KB	49 KB	693 KB	74%
	24	5 years		52 KB	771 KB	71%
	26	21 years		54 KB	823 KB	69%
8192	21	8 months	1.1 KB	53 KB	762 KB	85%
	24	5 years		57 KB	850 KB	84%
	26	21 years		59 KB	908 KB	83%

[a] Improvement over ρ signatures of Falcon-512 with signature size of 666 bytes.

Table 3. Space complexity of Squirrel.

6.2 Computational complexity and benchmarks

We implement our scheme and release the source code to the open domain[6]. We report benchmark result for the case of $\rho \in \{1024, 4096\}$ and $\tau = 21$; and give estimations for performance of $\tau = 24$ and 26. We run the benchmark over an AMD 5900x CPU with 12 cores, and with parallelization option turned on.

	$\rho = 1024$	$\rho = 4096$
$\mathcal{R}_{q_{\mathtt{HVC}}}$ NTT	4.1 μs	6.9 μs
$\mathcal{R}_{q_{\mathtt{HVC}}}$ NTT mul.	197 ns	260 ns
$\mathcal{R}_{q_{\mathtt{KOTS}}}$ NTT	5.8 μs	5.43 μs
$\mathcal{R}_{q_{\mathtt{KOTS}}}$ NTT mul.	508 ns	413 ns
ter-bin mul.	1.5 μs	
`HVC` hash	69 μs	107 μs
`KOTS` hash	111 μs	143 μs
gen randomizer	1.8 μs	
path randomization	274 μs	283 μs
1024 paths aggregation	680 ms	834 ms
1024 paths batch verifiction	20 ms	30 ms

Table 4. Microbenchmarks

Microbenchmarks We report the computation cost in Table 4. The main units of computations are

- A generic $\mathcal{R}_{q_{\mathtt{HVC}}}$ multiplication consists of converting both input polynomials into their NTT form ($O(n \log n)$), and conducting a coordinate-wise multiplication ($O(n)$), and convert the result back to integer polynomials. Denote this cost by c_1.
- A generic $\mathcal{R}_{q_{\mathtt{KOTS}}}$ multiplication consists of converting both input polynomials into their NTT form ($O(n \log n)$), and conducting a coordinate-wise multiplication ($O(n)$), and convert the result back to integer polynomials. Denote this cost by c_2.
- Multiply a binary polynomial with a fixed weight ternary polynomial. Denote this cost by c_3.

As examples, our hash function takes $2\lceil \log q_{\mathtt{HVC}} \rceil$ number of generic ring multiplications; randomizing a node takes $\lceil \log q_{\mathtt{HVC}} \rceil$ ternary ring multiplications. Concretely, our implementation reports that

- Hashing two child nodes into a parent node takes 107 microsecond;
- Hashing a KOTS public key into a leaf node takes 143 microsecond.

This is a lot better than $2 \log q$ number of multiplications due to a) parallelization, and b) the fact that hash parameters are already in the NTT form already; and that we only need to perform a single inverse NTT at the end.

Full Picture Similar to hash based signature schemes [BDH11, BHK+19], the key generation stage is the most expensive one in our case. It involves generating 2^τ KOTS keys, each costs 2γ generic ring multiplications; and the whole tree, at a cost of 2^τ node hashes and 2^τ leaf hashes. Overall cost is $2^\tau(2\lceil \log q_{\mathtt{HVC}} \rceil + 2\lceil \log q_{\mathtt{KOTS}} \rceil)c_1 + 2^{\tau+1}\gamma c_2 = 2^{\tau+1}((\lceil \log q_{\mathtt{HVC}} \rceil + \lceil \log q_{\mathtt{KOTS}} \rceil)c_1 + \gamma c_2)$.

[6] https://github.com/zhenfeizhang/squirrel

ρ	τ	Offline signing		Offline signing with cache		
		amortized	worst-case	$h = 12$	$h = 16$	$h = 20$
				2^{13} nodes	2^{17} nodes	2^{21} nodes
		$4\lceil \log q_{\mathtt{HVC}} \rceil c_1$	$2^{\tau+1}\lceil \log q_{\mathtt{HVC}} \rceil c_1$	$2^{\tau-h+3}\lceil \log q_{\mathtt{HVC}} \rceil c_1$		
1024	storage			7 MB	112 MB	1.8 GB
	21	41 μs	43 sec	42 ms	2.6 ms	164 μs
	24		6 min	336 ms	21 ms	1.3 ms
	26		23 min	1.3 sec	83 ms	5.2 ms
4096	storage			8 MB	128 MB	2 GB
	21	48 μs	52 sec	50 ms	3.1 ms	195 μs
	24		7 min	0.4 sec	25 ms	1.6 ms
	26		28 min	1.6 sec	99 ms	6.2 ms

Table 5. Estimated cost with cache

ρ	τ	Key Generation	Online signing	Aggregation*	Verification*
		$2^{\tau+1}((\lceil \log q_{\mathtt{HVC}} \rceil + \lceil \log q_{\mathtt{KOTS}} \rceil)c_1 + \gamma c_2)$	γc_2	$\rho c_2 + \rho(2\tau+2)\lceil \log q_{\mathtt{HVC}} \rceil c_3$	$2\lceil \log q_{\mathtt{HVC}} \rceil \tau c_1 + (2\gamma+\rho)c_2$
1024	21	4 min	2.1 ms	1.2 sec	19.5 ms
	24†	32 min		1.4 sec	22 ms
	26†	2 hour		1.5 sec	24 ms
4096	21	4.5 min	2.3 ms	1.4 sec	31 ms
	24†	36 min		1.6 sec	36 ms
	26†	2.4 hour		1.8 sec	38 ms

*: Aggregate and batch verify 1024 signatures. †: Estimations based on extrapolating $\tau = 21$ data.

Table 6. Benchmark results and estimations

Squirrel is an online/offline signature scheme. A speed sensitive signer may store the whole tree and avoid the entire offline phase. The online signing time becomes simply generating the OTS signature, which takes γ generic ring multiplications at a cost of γc_2. The signer will need to store the whole tree which consists of 2^τ nodes and 2^τ leaves, which translates into 5.1 gigabytes, 41 gigabytes and 164 gigabytes of data for each of the parameter settings respectively.

A space sensitive signer may store the last used path (and its adjacent nodes); and update it to its current path on-the-fly. Observe that any node will not be computed more than twice: the first time is during tree generation, and the second time is when it is firstly required in a path (and its adjacent nodes). Once a node is no longer required by a path nor the adjacent nodes, it will never be required again. Therefore, the amortized cost for each signatures will be 2 hashes (total number of nodes divided by total number of leaves). Since our hash function uses $2\lceil \log q_{\mathtt{HVC}} \rceil = 32$ generic ring multiplications, the amortized cost is 64 ring multiplications to update the path, and $\gamma = 44$ generic ring multiplications for KOTS signing.

In practice, the real bottleneck is the worst-case scenario, in which the signer will need to generate the signature for leaf with index $2^{\tau-1}$ (i.e., the first leaf of the second sub-tree) within a block interval. Concretely, the signer will need to generate $2^\tau - 2$ nodes, or equivalently, conduct $(2^{\tau+1} - 4)\lceil \log q_{\mathtt{HVC}} \rceil \approx 2^{\tau+1}\lceil \log q_{\mathtt{HVC}} \rceil$ generic ring multiplications. There are a few straightforward method to alleviate the situation. First, as an online/offline scheme, the signer always knows exactly when it will use leaf $2^{\tau-1}$. Therefore, it will be able to pre-compute this path offline. Secondly, if the signer is allowed some cache, it can store the top h levels of the tree, or $2^{h+1} - 2$ nodes, excluding the root. Accordingly, at the worst-case, the signer will need to online compute *two* sub-trees whose roots are the nodes at h-th level. That is $2(2^{\tau-h+1} - 1) \approx 2^{\tau-h+2}$ nodes, or $2^{\tau-h+3}\lceil \log q_{\mathtt{HVC}} \rceil$ generic ring multiplications in total. It also implies that the worst-case complexity will be reduced by half

for every additional level of nodes we cache. Table 6 gives a rough estimation of cache versus signing time.

To aggregate ρ signatures, the aggregator will need to multiply each path with some randomizers. There are $\rho(2\tau+2)$ number of nodes, leaves and KOTS public key nodes, combined; each requires $\lceil \log q_{\mathtt{HVC}} \rceil$ ternary ring multiplications. The aggregator will also need to randomize-then-aggregate KOTS signatures, which also incurs ρ generic ring multiplications. The total cost will be $\rho c_2 + \rho(2\tau+2)\lceil \log q_{\mathtt{HVC}} \rceil c_3$.

To verify an (aggregated) signature, the verifier will need to check that the path is valid with regard to the root of the tree. This takes $2\lceil \log q_{\mathtt{HVC}} \rceil \tau$ number of multiplications to check the path; and ρ number of multiplications to aggregate the public keys. In addition, the KOTS verification also uses 2γ ring multiplications.

6.3 Security estimation

Combinatorials First we need the randomizers to be sampled from a large space as per Lemma 2, i.e., $\binom{n}{\alpha}2^\alpha \geq 2^\lambda$. Setting $\alpha = 20$, i.e., the randomizer are sampled from the set of ternary polynomials with 20 non-zero entries, we have $\binom{n}{\alpha} \cdot 2^\alpha > 2^\lambda$.

Then, we discuss how we arrived to $\beta_{\mathtt{agg}} = 4096$. We assume that the aggregator may be malicious, that is, it can cherry pick their signatures so that, for a given node (i.e., an $\mathcal{R}_q$ element) for a given path, all the signatures will have 1s at a same index, Even so, the randomizers are outputs from the random oracle, where there are α number of ± 1s with equal probability. Therefore, for an aggregated polynomial, each coefficient can be seen as a sum of $\alpha\rho$ number of random elements in $\{-1, 1\}$. We need to set a bound $\beta_{\mathtt{agg}}$ such that, the probability that *all* coefficients for *all* nodes are bounded by $\beta_{\mathtt{agg}}$ in absolute value with overwhelming probability, i.e.,

$$2\tau\lceil \log q_{\mathtt{HVC}} \rceil n \left(\Pr\left[\forall i \in [\alpha\rho], b_i \leftarrow \{-1,1\} : \left| \sum_{i=1}^{\alpha\rho} b_i \right| \geq \beta_{\mathtt{agg}} \right] \right) \leq 2^{-\lambda}$$

For $\alpha = 20$ we are able to set $\beta_{\mathtt{agg}} = 4096$. Additionally, we require that $2\beta_{\mathtt{agg}} < q_{\mathtt{HVC}}/2$ so that in Lemma 9 the extracted vector is indeed a short solution to the SIS problem.

The messages are hashed into $\mathcal{T}_{\beta_s}$. Therefore, we need to set $\beta_s = 44$ so that $|\mathcal{T}_{\beta_s}| > 2^{2\lambda}$. Note that we need $(4\rho+8)\alpha\beta_s < q_{\mathtt{KOTS}}/2$ so that in Lemma 17 the extracted vector is indeed a short solution to the SIS problem. Per Lemma 17, we then need to set $\gamma = 44$ such that $2^{(3\lambda+1)/n\gamma} \cdot q_{\mathtt{KOTS}}^{1/\gamma} \leq 3/2$.

Lattice attacks For a root Hermite factor $c \leq 1.005$, the LWE-estimator [APS15] reported that BKZ [CN11] will be able to find a short vector for a block size of 286. Such a lattice reduction requires 112 bits operations under the *realistic model* in [ADPS16], which estimates the SVP cost from [BDGL16]. For a BKZ of block size β, the cost in this model is estimated by $2^{0.292\beta+16.4+\log(\#\mathtt{SVP\ calls})}$. This consists of the number of operations in a single sieving ($2^{0.292\beta}$), a constant factor from experiments ($2^{16.4}$) attributed to per operation cost, and the number of SVP calls. Note that [ADPS16] also proposed a *core-sieving-SVP* model that ignores all the constant factors ($2^{16.4}$ per operations, and the number of svp calls). We do not adopt this model.

Our HVC scheme requires that the $\mathsf{SIS}_{\mathcal{R},q,2\lceil \log q_{\mathtt{HVC}} \rceil, 4\rho\alpha}$ problem is hard as per Theorem 5 and Lemma 9, for $q_{\mathtt{HVC}} = \mathtt{61441}$, $\rho \in [4096]$ and $\alpha = 20$. An SIS becomes easier when the target solution

is longer, therefore it is sufficient to analyze the case $\rho = 4096$. This instantiation yields a lattice of dimension $(2\lceil \log q_{\mathtt{HVC}} \rceil + 1)n$ and determinant $q_{\mathtt{HVC}}^n$. As per [MR09], a lattice reduction algorithm will find a short vector of $2^{2\sqrt{n \log q_{\mathtt{HVC}} \log c}}$ for some root Hermite factor c that depends on the lattice reduction algorithm. In the meantime, the vector we are searching for has an infinity norm of $\beta_{\mathtt{agg}}$, which means its ℓ_2 norm is bounded by $t_{\mathtt{HVC}} = \sqrt{2n \log q_{\mathtt{HVC}}}\beta_{\mathtt{agg}}$. With our choice of parameters, a lattice reduction algorithm will be able to find this target vector for $c < 1.005$.

Last, we analysis the hardness of the $\mathsf{SIS}_{\mathcal{R},q_{\mathtt{KOTS}},\gamma,(4\rho+8)\alpha\beta_s}$ assumption for our KOTS scheme as per Lemma 17. This follows a similar analysis as the above SIS analysis. Here, we have a lattice of dimension $(\gamma+1)n$ and determinant $q_{\mathtt{KOTS}}^n$. An aggregated signature has an infinity norm bound of $(4\rho+8)\alpha\beta_s$, which implies $t_{\mathtt{HOST}} = \sqrt{\gamma n}(4\rho+8)\alpha\beta_s$ in ℓ_2 norm. We also require that $t_{\mathtt{HOST}} < c^{\dim} 2^{2\sqrt{n \log q_{\mathtt{KOTS}} \log c}}$ so that BKZ cannot solve this instance of SIS problem. With our parameter sets we have $c < 1.004$.

References

ADPS16. Erdem Alkim, Léo Ducas, Thomas Pöppelmann, and Peter Schwabe. Post-quantum key exchange - A new hope. In Thorsten Holz and Stefan Savage, editors, *USENIX Security 2016: 25th USENIX Security Symposium*, pages 327–343, Austin, TX, USA, August 10–12, 2016. USENIX Association. 6.3

AGH10. Jae Hyun Ahn, Matthew Green, and Susan Hohenberger. Synchronized aggregate signatures: new definitions, constructions and applications. In Ehab Al-Shaer, Angelos D. Keromytis, and Vitaly Shmatikov, editors, *ACM CCS 2010: 17th Conference on Computer and Communications Security*, pages 473–484, Chicago, Illinois, USA, October 4–8, 2010. ACM Press. 1.1

Ajt99. Miklós Ajtai. Generating hard instances of the short basis problem. In Jirí Wiedermann, Peter van Emde Boas, and Mogens Nielsen, editors, *ICALP 99: 26th International Colloquium on Automata, Languages and Programming*, volume 1644 of *Lecture Notes in Computer Science*, pages 1–9, Prague, Czech Republic, July 11–15, 1999. Springer, Heidelberg, Germany. 1.4

APS15. Martin R. Albrecht, Rachel Player, and Sam Scott. On the concrete hardness of learning with errors. Cryptology ePrint Archive, Report 2015/046, 2015. https://eprint.iacr.org/2015/046. 6.3

BCJ08. Ali Bagherzandi, Jung Hee Cheon, and Stanislaw Jarecki. Multisignatures secure under the discrete logarithm assumption and a generalized forking lemma. In Peng Ning, Paul F. Syverson, and Somesh Jha, editors, *ACM CCS 2008: 15th Conference on Computer and Communications Security*, pages 449–458, Alexandria, Virginia, USA, October 27–31, 2008. ACM Press. 1

BDD+00. Mike Burmester, Yvo Desmedt, Hiroshi Doi, Masahiro Mambo, Eiji Okamoto, Mitsuru Tada, and Yuko Yoshifuji. A structured ElGamal-type multisignature scheme. In Hideki Imai and Yuliang Zheng, editors, *PKC 2000: 3rd International Workshop on Theory and Practice in Public Key Cryptography*, volume 1751 of *Lecture Notes in Computer Science*, pages 466–483, Melbourne, Victoria, Australia, January 18–20, 2000. Springer, Heidelberg, Germany. 1

BDF+11. Dan Boneh, Özgür Dagdelen, Marc Fischlin, Anja Lehmann, Christian Schaffner, and Mark Zhandry. Random oracles in a quantum world. In Dong Hoon Lee and Xiaoyun Wang, editors, *Advances in Cryptology – ASIACRYPT 2011*, volume 7073 of *Lecture Notes in Computer Science*, pages 41–69, Seoul, South Korea, December 4–8, 2011. Springer, Heidelberg, Germany. 1.3

BDGL16. Anja Becker, Léo Ducas, Nicolas Gama, and Thijs Laarhoven. New directions in nearest neighbor searching with applications to lattice sieving. In Robert Krauthgamer, editor, *27th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 10–24, Arlington, VA, USA, January 10–12, 2016. ACM-SIAM. 6.3

BDH11. Johannes A. Buchmann, Erik Dahmen, and Andreas Hülsing. XMSS - A practical forward secure signature scheme based on minimal security assumptions. In Bo-Yin Yang, editor, *Post-Quantum Cryptography - 4th International Workshop, PQCrypto 2011*, pages 117–129, Tapei, Taiwan, November 29 – December 2 2011. Springer, Heidelberg, Germany. 6.2

BDN18. Dan Boneh, Manu Drijvers, and Gregory Neven. Compact multi-signatures for smaller blockchains. In Thomas Peyrin and Steven Galbraith, editors, *Advances in Cryptology – ASIACRYPT 2018, Part II*, volume 11273 of *Lecture Notes in Computer Science*, pages 435–464, Brisbane, Queensland, Australia, December 2–6, 2018. Springer, Heidelberg, Germany. 1

BGLS02. Dan Boneh, Craig Gentry, Ben Lynn, and Hovav Shacham. Aggregate and verifiably encrypted signatures from bilinear maps. Cryptology ePrint Archive, Report 2002/175, 2002. https://eprint.iacr.org/2002/175. 1

BHK+19. Daniel J. Bernstein, Andreas Hülsing, Stefan Kölbl, Ruben Niederhagen, Joost Rijneveld, and Peter Schwabe. The SPHINCS+ signature framework. In Lorenzo Cavallaro, Johannes Kinder, XiaoFeng Wang, and Jonathan Katz, editors, *ACM CCS 2019: 26th Conference on Computer and Communications Security*, pages 2129–2146. ACM Press, November 11–15, 2019. 6.2

BK20. Dan Boneh and Sam Kim. One-time and interactive aggregate signatures from lattices. https://crypto.stanford.edu/~skim13/agg_ots.pdf, 2020. 4, 4

BLS01. Dan Boneh, Ben Lynn, and Hovav Shacham. Short signatures from the Weil pairing. In Colin Boyd, editor, *Advances in Cryptology – ASIACRYPT 2001*, volume 2248 of *Lecture Notes in Computer Science*, pages 514–532, Gold Coast, Australia, December 9–13, 2001. Springer, Heidelberg, Germany. 1.2

BN06. Mihir Bellare and Gregory Neven. Multi-signatures in the plain public-key model and a general forking lemma. In Ari Juels, Rebecca N. Wright, and Sabrina De Capitani di Vimercati, editors, *ACM CCS 2006: 13th Conference on Computer and Communications Security*, pages 390–399, Alexandria, Virginia, USA, October 30 – November 3, 2006. ACM Press. 1, 1.3, 2

Bol03. Alexandra Boldyreva. Threshold signatures, multisignatures and blind signatures based on the gap-Diffie-Hellman-group signature scheme. In Yvo Desmedt, editor, *PKC 2003: 6th International Workshop on Theory and Practice in Public Key Cryptography*, volume 2567 of *Lecture Notes in Computer Science*, pages 31–46, Miami, FL, USA, January 6–8, 2003. Springer, Heidelberg, Germany. 1

BTT22. Cecilia Boschini, Akira Takahashi, and Mehdi Tibouchi. Musig-l: Lattice-based multi-signature with single-round online phase. In Yevgeniy Dodis and Tom Shrimpton, editors, *Advances in Cryptology – CRYPTO 2022*, Lecture Notes in Computer Science, Santa Barbara, CA, USA, August 2022. Springer, Heidelberg, Germany. 1

CN11. Yuanmi Chen and Phong Q. Nguyen. BKZ 2.0: Better lattice security estimates. In Dong Hoon Lee and Xiaoyun Wang, editors, *Advances in Cryptology – ASIACRYPT 2011*, volume 7073 of *Lecture Notes in Computer Science*, pages 1–20, Seoul, South Korea, December 4–8, 2011. Springer, Heidelberg, Germany. 6.3

DGNW20. Manu Drijvers, Sergey Gorbunov, Gregory Neven, and Hoeteck Wee. Pixel: Multi-signatures for consensus. In Srdjan Capkun and Franziska Roesner, editors, *USENIX Security 2020: 29th USENIX Security Symposium*, pages 2093–2110. USENIX Association, August 12–14, 2020. 1, 1.1

DKL+18. Léo Ducas, Eike Kiltz, Tancrède Lepoint, Vadim Lyubashevsky, Peter Schwabe, Gregor Seiler, and Damien Stehlé. CRYSTALS-Dilithium: A lattice-based digital signature scheme. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2018(1):238–268, 2018. https://tches.iacr.org/index.php/TCHES/article/view/839. 1.1

DOTT21. Ivan Damgård, Claudio Orlandi, Akira Takahashi, and Mehdi Tibouchi. Two-round n-out-of-n and multi-signatures and trapdoor commitment from lattices. In Juan Garay, editor, *PKC 2021: 24th International Conference on Theory and Practice of Public Key Cryptography, Part I*, volume 12710 of *Lecture Notes in Computer Science*, pages 99–130, Virtual Event, May 10–13, 2021. Springer, Heidelberg, Germany. 1

EGM90. Shimon Even, Oded Goldreich, and Silvio Micali. On-line/off-line digital schemes. In Gilles Brassard, editor, *Advances in Cryptology – CRYPTO'89*, volume 435 of *Lecture Notes in Computer Science*, pages 263–275, Santa Barbara, CA, USA, August 20–24, 1990. Springer, Heidelberg, Germany. 1.1

ES16. Rachid El Bansarkhani and Jan Sturm. An efficient lattice-based multisignature scheme with applications to bitcoins. In Sara Foresti and Giuseppe Persiano, editors, *CANS 16: 15th International Conference on Cryptology and Network Security*, volume 10052 of *Lecture Notes in Computer Science*, pages 140–155, Milan, Italy, November 14–16, 2016. Springer, Heidelberg, Germany. 1

Eth22. EtherScan. Etherscan.io. https://etherscan.io/nodetracker, 2022. 6

FH19. Masayuki Fukumitsu and Shingo Hasegawa. A tightly-secure lattice-based multisignature. In *6th ASIA Public-Key Cryptography Workshop*, page 3–11, Auckland, New Zealand, 2019. Association for Computing Machinery. 1

FH20. Masayuki Fukumitsu and Shingo Hasegawa. A lattice-based provably secure multisignature scheme in quantum random oracle model. In Khoa Nguyen, Wenling Wu, Kwok-Yan Lam, and Huaxiong Wang, editors, *ProvSec 2020: 14th International Conference on Provable Security*, volume 12505 of *Lecture Notes in Computer Science*, pages 45–64, Singapore, November 29 – December 1, 2020. Springer, Heidelberg, Germany. 1

GR06. Craig Gentry and Zulfikar Ramzan. Identity-based aggregate signatures. In Moti Yung, Yevgeniy Dodis, Aggelos Kiayias, and Tal Malkin, editors, *PKC 2006: 9th International Conference on Theory and Practice*

of Public Key Cryptography, volume 3958 of *Lecture Notes in Computer Science*, pages 257–273, New York, NY, USA, April 24–26, 2006. Springer, Heidelberg, Germany. 1.1

HW18. Susan Hohenberger and Brent Waters. Synchronized aggregate signatures from the RSA assumption. In Jesper Buus Nielsen and Vincent Rijmen, editors, *Advances in Cryptology – EUROCRYPT 2018, Part II*, volume 10821 of *Lecture Notes in Computer Science*, pages 197–229, Tel Aviv, Israel, April 29 – May 3, 2018. Springer, Heidelberg, Germany. 1.1

IN83. Kazuharu Itakura and Katsuhiro Nakamura. A public-key cryptosystem suitable for digital multisignatures. *NEC Research & Development*, (71):1–8, 1983. 1

KD20. Meenakshi Kansal and Ratna Dutta. Round optimal secure multisignature schemes from lattice with public key aggregation and signature compression. In Abderrahmane Nitaj and Amr M. Youssef, editors, *AFRICACRYPT 20: 12th International Conference on Cryptology in Africa*, volume 12174 of *Lecture Notes in Computer Science*, pages 281–300, Cairo, Egypt, July 20–22, 2020. Springer, Heidelberg, Germany. 1

Lib22. LibSecP. libsecp256k1: Optimized c library for ecdsa signatures and secret/public key operations on curve secp256k1. https://github.com/bitcoin-core/secp256k1, 2022. 1.1

LLNW16. Benoît Libert, San Ling, Khoa Nguyen, and Huaxiong Wang. Zero-knowledge arguments for lattice-based accumulators: Logarithmic-size ring signatures and group signatures without trapdoors. In Marc Fischlin and Jean-Sébastien Coron, editors, *Advances in Cryptology – EUROCRYPT 2016, Part II*, volume 9666 of *Lecture Notes in Computer Science*, pages 1–31, Vienna, Austria, May 8–12, 2016. Springer, Heidelberg, Germany. 1.4, 3.1

LM08. Vadim Lyubashevsky and Daniele Micciancio. Asymptotically efficient lattice-based digital signatures. In Ran Canetti, editor, *TCC 2008: 5th Theory of Cryptography Conference*, volume 4948 of *Lecture Notes in Computer Science*, pages 37–54, San Francisco, CA, USA, March 19–21, 2008. Springer, Heidelberg, Germany. 4, 4

LOS+06. Steve Lu, Rafail Ostrovsky, Amit Sahai, Hovav Shacham, and Brent Waters. Sequential aggregate signatures and multisignatures without random oracles. In Serge Vaudenay, editor, *Advances in Cryptology – EUROCRYPT 2006*, volume 4004 of *Lecture Notes in Computer Science*, pages 465–485, St. Petersburg, Russia, May 28 – June 1, 2006. Springer, Heidelberg, Germany. 1

LTT20. Zi-Yuan Liu, Yi-Fan Tseng, and Raylin Tso. Cryptanalysis of a round optimal lattice-based multisignature scheme. Cryptology ePrint Archive, Report 2020/1172, 2020. https://eprint.iacr.org/2020/1172. 1

MEJ20. Soo Hoon Maeng, Meryam Essaid, and Hongtaek Ju. Analysis of ethereum network properties and behavior of influential nodes. In *21st Asia-Pacific Network Operations and Management Symposium*, pages 203–207, Daegu, South Korea, September 2020. IEEE. 6

Mic07. Daniele Micciancio. Generalized compact knapsacks, cyclic lattices, and efficient one-way functions. *computatinal complexity*, 16(4):365–411, December 2007. 1

MJ19. Changshe Ma and Mei Jiang. Practical lattice-based multisignature schemes for blockchains. *IEEE Access*, 7:179765–179778, 2019. 1

MOR01. Silvio Micali, Kazuo Ohta, and Leonid Reyzin. Accountable-subgroup multisignatures: Extended abstract. In Michael K. Reiter and Pierangela Samarati, editors, *ACM CCS 2001: 8th Conference on Computer and Communications Security*, pages 245–254, Philadelphia, PA, USA, November 5–8, 2001. ACM Press. 1

MR09. Daniele Micciancio and Oded Regev. Lattice-based cryptography. In Daniel J. Bernstein, Jonhannes Buchmann, and Erik Dahmen, editors, *Post-quantum Cryptography*, chapter 5, pages 147–191. Springer, Heidelberg, Germany, Berlin, Heidelberg, 2009. 6.3

NRS21. Jonas Nick, Tim Ruffing, and Yannick Seurin. MuSig2: Simple two-round Schnorr multi-signatures. In Tal Malkin and Chris Peikert, editors, *Advances in Cryptology – CRYPTO 2021, Part I*, volume 12825 of *Lecture Notes in Computer Science*, pages 189–221, Virtual Event, August 16–20, 2021. Springer, Heidelberg, Germany. 1

OO93. Kazuo Ohta and Tatsuaki Okamoto. A digital multisignature scheme based on the Fiat-Shamir scheme. In Hideki Imai, Ronald L. Rivest, and Tsutomu Matsumoto, editors, *Advances in Cryptology – ASIACRYPT'91*, volume 739 of *Lecture Notes in Computer Science*, pages 139–148, Fujiyoshida, Japan, November 11–14, 1993. Springer, Heidelberg, Germany. 1

PD20. Chunyan Peng and Xiujuan Du. New lattice-based digital multi-signature scheme. In *6th International Conference of Pioneering Computer Scientists, Engineers and Educators*, volume 1258 of *CCIS*, pages 129–137, Taiyuan, China, September 2020. Springer, Heidelberg, Germany. 1

PFH+20. Thomas Prest, Pierre-Alain Fouque, Jeffrey Hoffstein, Paul Kirchner, Vadim Lyubashevsky, Thomas Pornin, Thomas Ricosset, Gregor Seiler, William Whyte, and Zhenfei Zhang. FALCON. Technical report,

National Institute of Standards and Technology, 2020. available at https://csrc.nist.gov/projects/post-quantum-cryptography/round-3-submissions. 1.1

PS96. David Pointcheval and Jacques Stern. Security proofs for signature schemes. In Ueli M. Maurer, editor, *Advances in Cryptology – EUROCRYPT'96*, volume 1070 of *Lecture Notes in Computer Science*, pages 387–398, Saragossa, Spain, May 12–16, 1996. Springer, Heidelberg, Germany. 1.3

PSTY13. Charalampos Papamanthou, Elaine Shi, Roberto Tamassia, and Ke Yi. Streaming authenticated data structures. In Thomas Johansson and Phong Q. Nguyen, editors, *Advances in Cryptology – EUROCRYPT 2013*, volume 7881 of *Lecture Notes in Computer Science*, pages 353–370, Athens, Greece, May 26–30, 2013. Springer, Heidelberg, Germany. 1.4, 3.1

Sho94. Peter W. Shor. Algorithms for quantum computation: Discrete logarithms and factoring. In *35th Annual Symposium on Foundations of Computer Science*, pages 124–134, Santa Fe, NM, USA, November 20–22, 1994. IEEE Computer Society Press. 1

Sup22. Supranational. blst: A bls12-381 signature library focused on performance and security. https://github.com/supranational/blst, 2022. 1.1

YZ21. Takashi Yamakawa and Mark Zhandry. Classical vs quantum random oracles. In Anne Canteaut and François-Xavier Standaert, editors, *Advances in Cryptology – EUROCRYPT 2021, Part II*, volume 12697 of *Lecture Notes in Computer Science*, pages 568–597, Zagreb, Croatia, October 17–21, 2021. Springer, Heidelberg, Germany. 1.3